Introduction to Agentic AI

-- Fine-Tuning

Instructor: Guangjing Wang

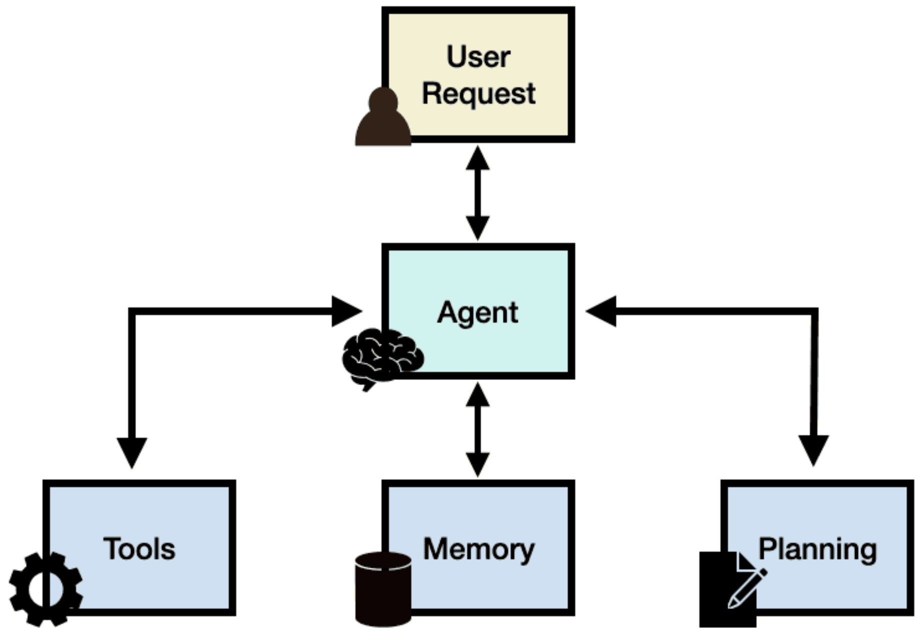
guangjingwang@usf.edu

Last Lecture

- Masked language modeling
- Causal language modeling
- Sequence to sequence modeling
- LLM inference process
- Evaluation and Optimization Inference

Questions

- The base model is only trained to predict the next token.
- Why can large language model think/plan?
- Why can large language model call tools (other function)?

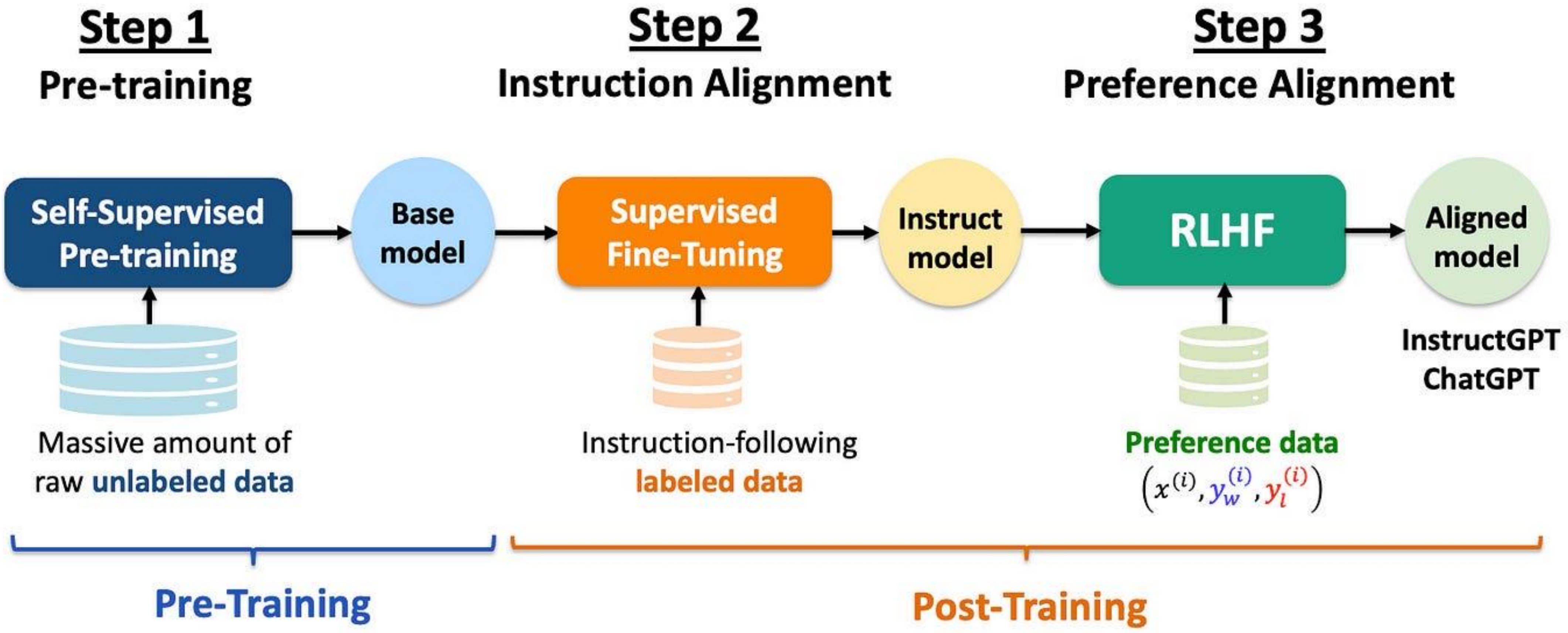


https://www.promptingguide.ai/research/llm-agents

This Lecture

- Instruction tuning
  - Chat Template
  - SFT
  - LoRa
- Preference alignment
  - RLHF
  - DPO

Model Training Pipeline



https://youssefh.substack.com/p/visual-guide-to-llm-preference-tuning

Base Model and Instruct Model

- **Base model**: Trained on raw text to predict the next token.
- **Instruct Model:** Fine-tuned to follow instructions and engage in conversations.
- **From base to instruction:**
  - **Supervised fine-tuning**: The technique used to train the model to generate appropriate responses.
  - **Chat template**: A structured format for interactions between language models, users, and external tools, designed for fine-tuning dataset.

Supervised Fine-tuning (SFT): Instruction Tuning

- Instruction tuning is the process of adapting pre-trained language models to follow human instructions and engage in conversations.
  - Follow user instructions accurately
  - Engage in natural conversations
  - Provide helpful, harmless and honest responses
  - Maintain context across multi-turn interaction
  - Use tools to perform tasks
- This is achieved through supervised fine-tuning on carefully curated dataset!

Chat Templates

- A consistent format for structuring interactions between language models, users, and external tools
- Think Chat Templates as the “grammar” that teaches models how to understand conversations, distinguish between different speakers, and respond appropriately
- Chat Template Playground: https://huggingface.co/spaces/huggingfacejs/chat-template-playground

Chat Templates

ChatML (Chat Markup Language)

- Uses **special tokens**

  <|im_start|> and <|im_end|> to mark message boundaries
- Simple **role-based structure** (system, user, assistant)
- **Thinking**/reasoning wrapped in <think>...</think> blocks
- **Tool** definitions and calls use XML-style tags

Harmony (OpenAI)

- **Multi-channel architecture**: Messages can be tagged as analysis (reasoning), commentary (tool preambles), or final (user-facing)
- **Role hierarchy**: system > developer > user > assistant > tool (for handling instruction conflicts)
- **Message routing**: Uses to= syntax for directing messages between components
- **TypeScript-style tool definitions**: More expressive and concise than JSON schemas

ChatML (Chat Markup Language)

Role: Identifies who is speaking

- System Messages: Set behavior and context for the entire conversation
- User Messages: Questions, requests, or statements from the human user
- Assistant Messages: Responses from the AI model
- Tool Messages: Results from function calls (for advanced use cases)

Content: The actual message content.

ChatML Template Example

```
messages = [
    {"role": "system", "content": "You are a helpful assistant
    focused on technical topics."},
    {"role": "user", "content": "Can you explain what a chat
    template is?"},
    {"role": "assistant", "content": "A chat template structures
    conversations between users and AI models by providing a
    consistent format that helps the model understand different
    roles and maintain context."}
]
```

```
<|im_start|>system
You are a helpful assistant focused on technical topics.<|im_end|>
<|im_start|>user
Can you explain what a chat template is?<|im_end|>
<|im_start|>assistant
A chat template structures conversations between users and AI
models by providing a consistent format that helps the model
understand different roles and maintain context.<|im_end|>
<|im_start|>assistant
```

Generation Prompts in ChatML Template

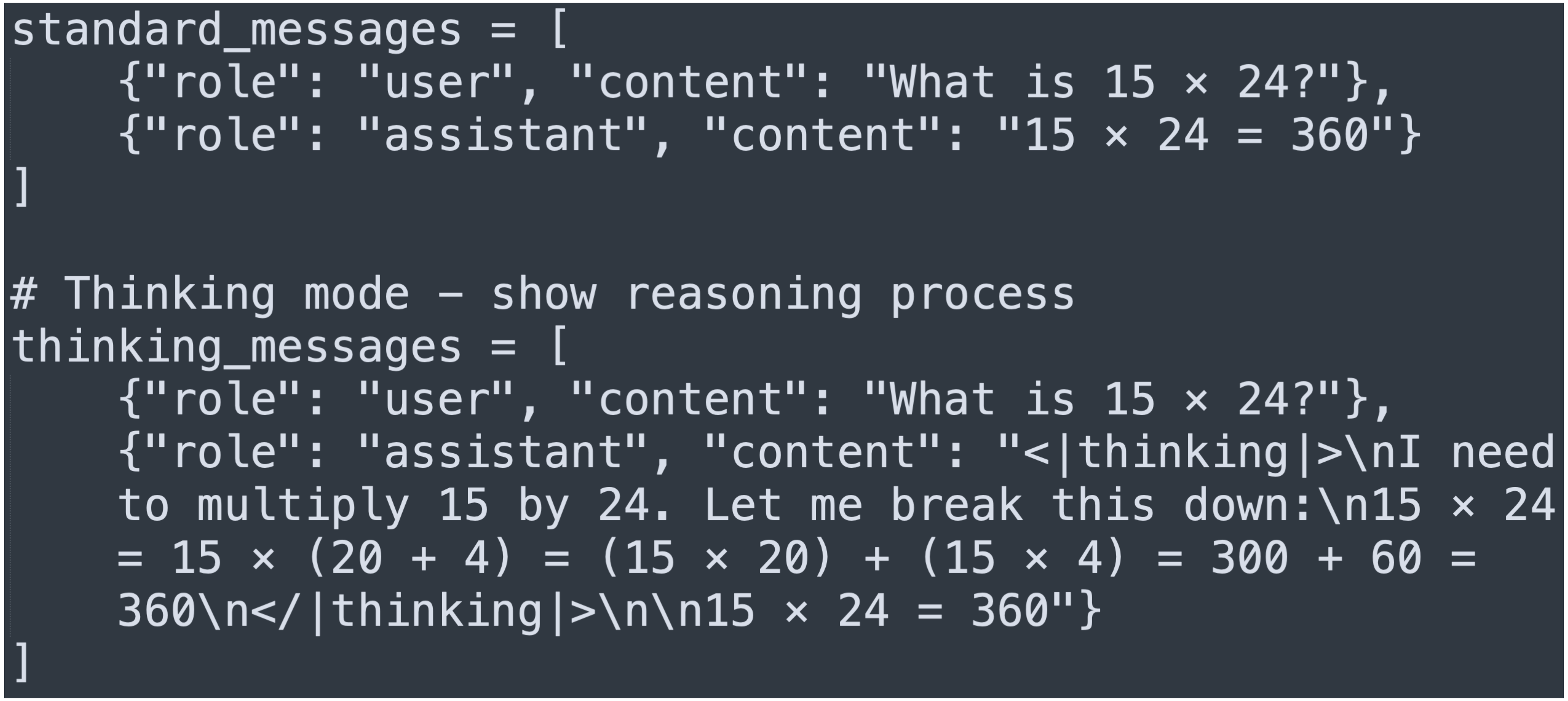
```
<|im_start|>user
Hi there!<|im_end|>
<|im_start|>assistant
Nice to meet you!<|im_end|>
<|im_start|>user
Can I ask a question?<|im_end|>
```

Doing something unexpected like continuing the user’s message

```
<|im_start|>user
Hi there!<|im_end|>
<|im_start|>assistant
Nice to meet you!<|im_end|>
<|im_start|>user
Can I ask a question?<|im_end|>
<|im_start|>assistant
```

Write a bot response

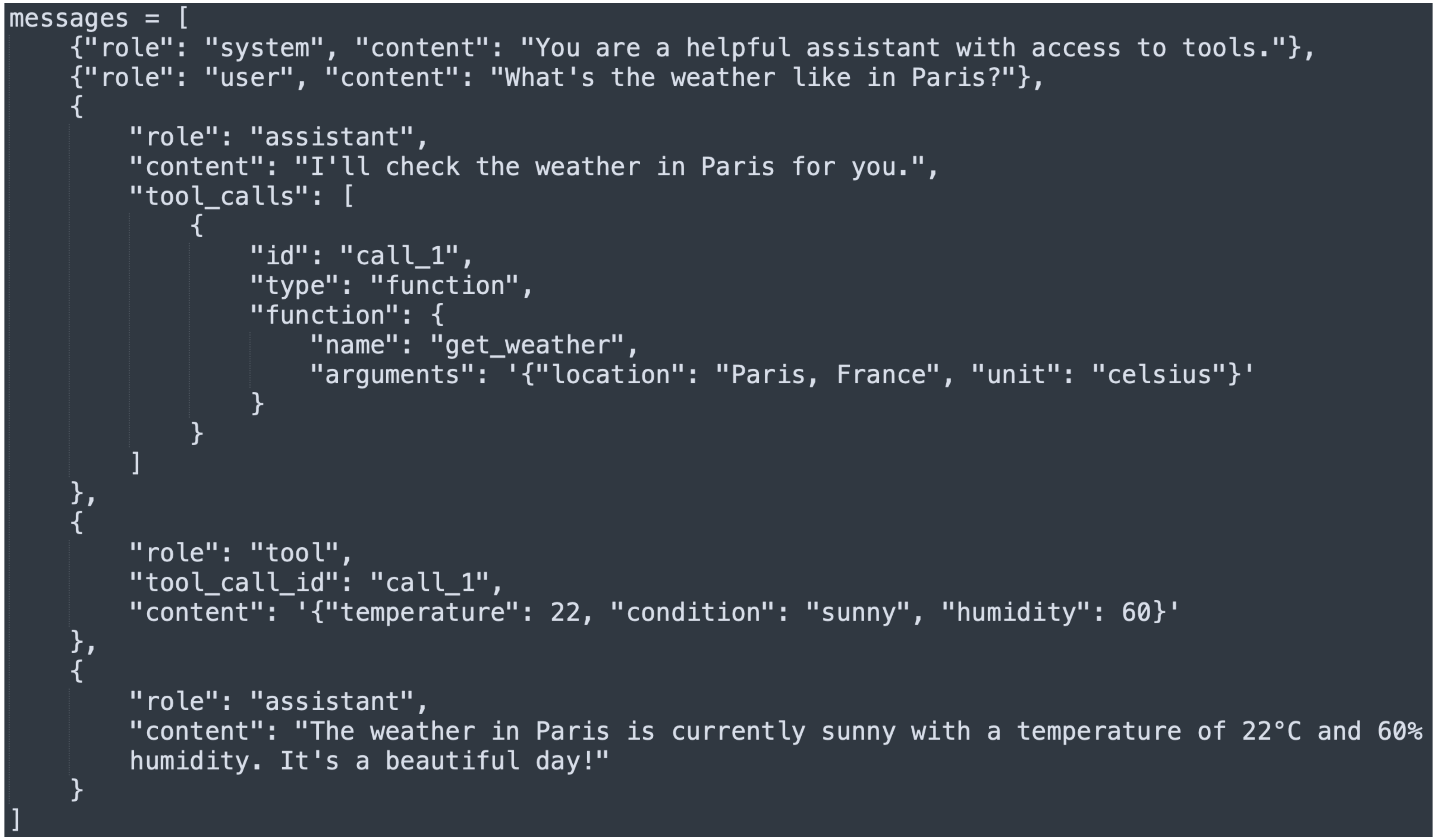
Chat Example with Thinking in Training Data



```
standard_messages = [
    {"role": "user", "content": "What is 15 × 24?"},
    {"role": "assistant", "content": "15 × 24 = 360"}
]

# Thinking mode - show reasoning process
thinking_messages = [
    {"role": "user", "content": "What is 15 × 24?"},
    {"role": "assistant", "content": "<|thinking|>\nI need
    to multiply 15 by 24. Let me break this down:\n15 × 24
    = 15 × (20 + 4) = (15 × 20) + (15 × 4) = 300 + 60 =
    360\n</|thinking|>\n\n15 × 24 = 360"}
]
```

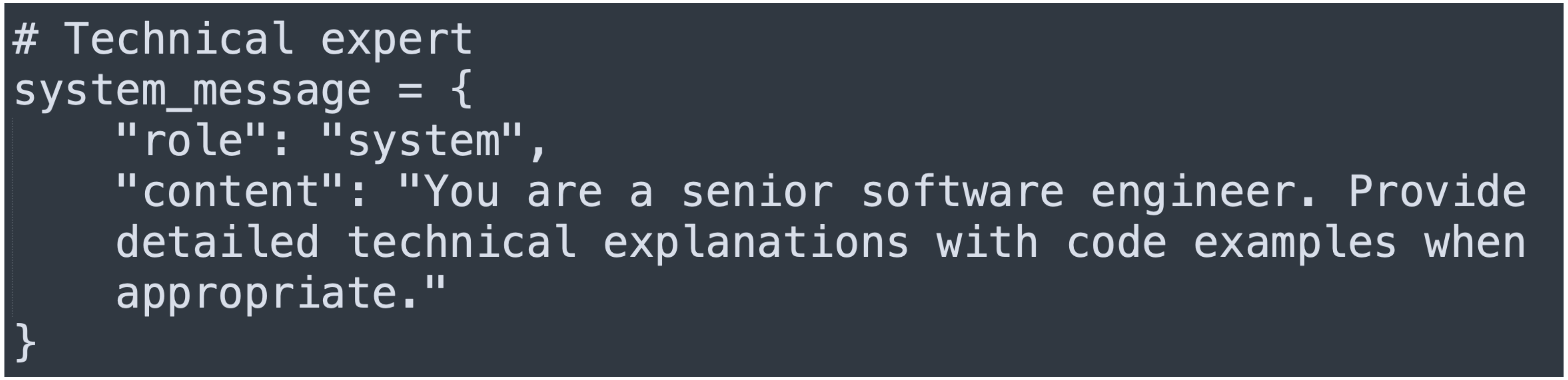
Chat Example with Tool in Training Data



```
messages = [
    {"role": "system", "content": "You are a helpful assistant with access to tools."},
    {"role": "user", "content": "What's the weather like in Paris?"},
    {
        "role": "assistant",
        "content": "I'll check the weather in Paris for you.",
        "tool_calls": [
            {
                "id": "call_1",
                "type": "function",
                "function": {
                    "name": "get_weather",
                    "arguments": '{"location": "Paris, France", "unit": "celsius"}'
                }
            }
        ]
    },
    {
        "role": "tool",
        "tool_call_id": "call_1",
        "content": '{"temperature": 22, "condition": "sunny", "humidity": 60}'
    },
    {
        "role": "assistant",
        "content": "The weather in Paris is currently sunny with a temperature of 22°C and 60%
        humidity. It's a beautiful day!"
    }
]
```

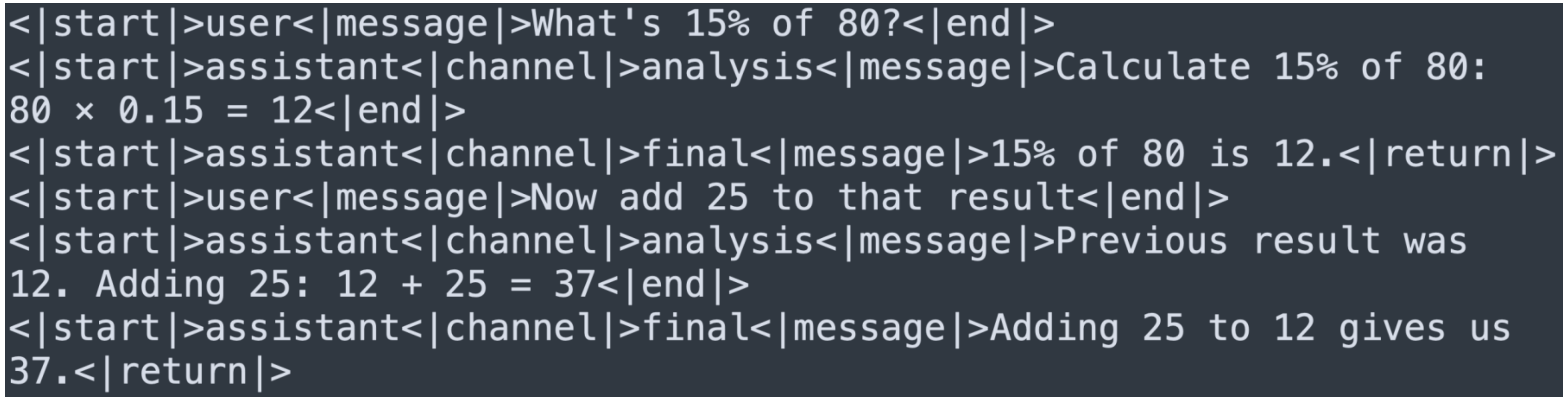
System Message

- Acting as persistent instructions that influence all subsequent interactions;
- System messages are the first message in the conversation, and they set the tone for the entire conversation. They should be specific, set boundaries, provide context, and use examples.

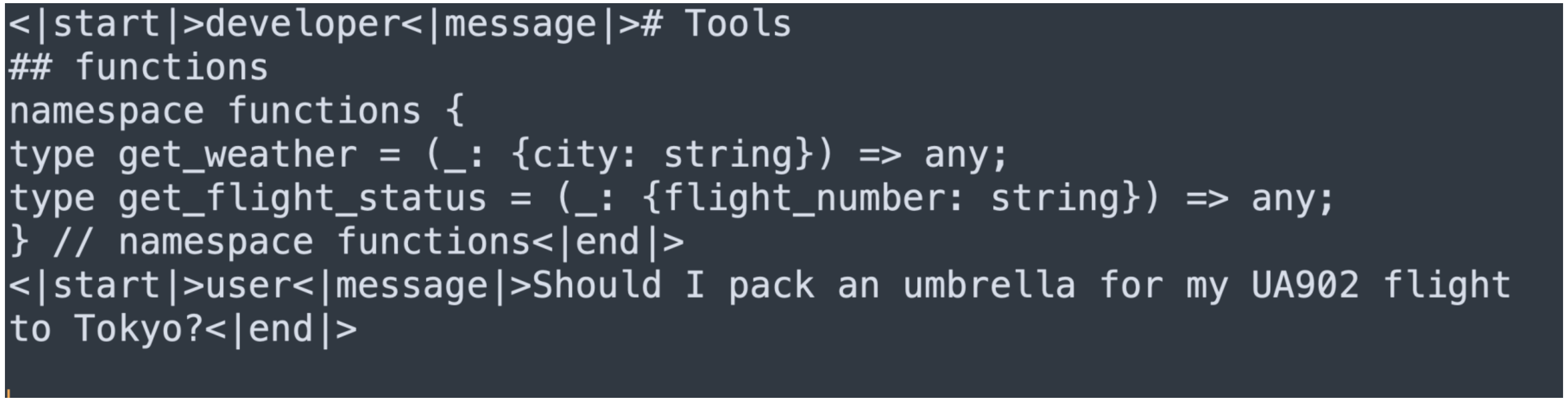


```
# Technical expert
system_message = {
    "role": "system",
    "content": "You are a senior software engineer. Provide
    detailed technical explanations with code examples when
    appropriate."
}
```

OpenAI Harmony Template Example



```
<|start|>user<|message|>What's 15% of 80?<|end|>
<|start|>assistant<|channel|>analysis<|message|>Calculate 15% of 80:
80 × 0.15 = 12<|end|>
<|start|>assistant<|channel|>final<|message|>15% of 80 is 12.<|return|>
<|start|>user<|message|>Now add 25 to that result<|end|>
<|start|>assistant<|channel|>analysis<|message|>Previous result was
12. Adding 25: 12 + 25 = 37<|end|>
<|start|>assistant<|channel|>final<|message|>Adding 25 to 12 gives us
37.<|return|>
```



```
<|start|>developer<|message|># Tools
## functions
namespace functions {
type get_weather = (_: {city: string}) => any;
type get_flight_status = (_: {flight_number: string}) => any;
} // namespace functions<|end|>
<|start|>user<|message|>Should I pack an umbrella for my UA902 flight
to Tokyo?<|end|>
```

https://huggingface.co/blog/kuotient/chatml-vs-harmony

Pitfalls about Fine-tuning Dataset and Inference with Chat Template

- **Template mismatch**: Using a different template than the model was fine-tuned on.
- **Double special tokens**: Adding special tokens when the template already includes them.
- **Missing system messages**: Not providing enough context for consistent model behavior.
- **Inconsistent formatting**: Mixing different conversation formats in the same dataset.
- **Ignoring tool syntax**: Not properly formatting tool calls and responses.
- **Context overflow**: Not managing long conversations within token limits.

Why SFT Works: The Science Behind It

- **Behavioral Adaptation:** updating the attention mechanisms to focus on instruction cues in language and adjusting the output distribution to favor the desired responses.
- **Task Specialization:** SFT teaches the model how to format and present this knowledge appropriately.
- **Safety Alignment:** Through exposure to carefully curated examples, the model learns to be more helpful and harmless. This involves both learning what to say and what not to say in various situations.

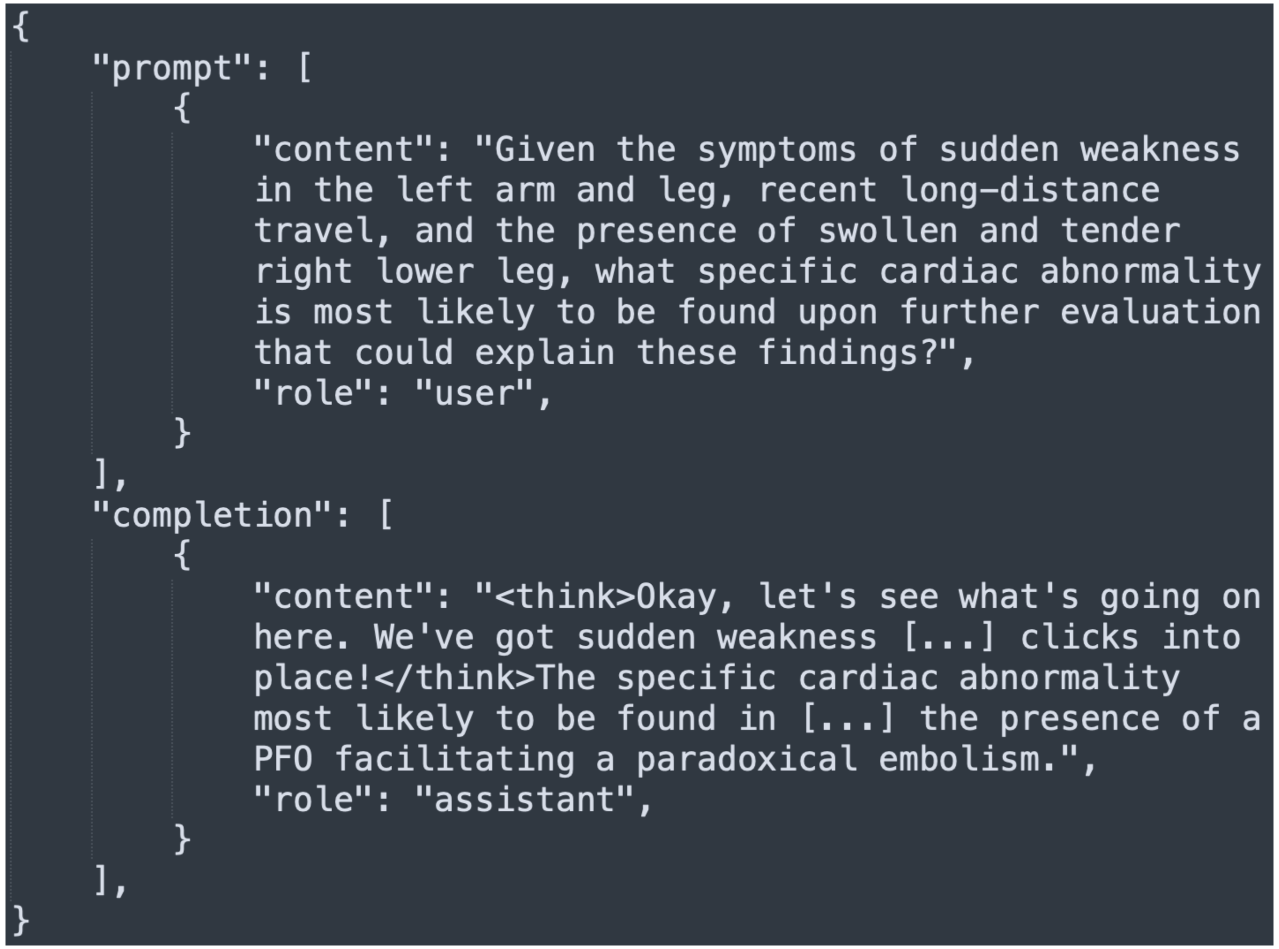
Is SFT Appropriate for Your Project?

- Have you tried prompt engineering with existing instruction-tuned models?
- Do you need consistent output formats that prompting cannot achieve?
- Is your domain specialized enough that general models struggle?
- Do you have high-quality training data (at least 1,000 examples)?
- Do you have the computational resources for training and evaluation?

The SFT Process

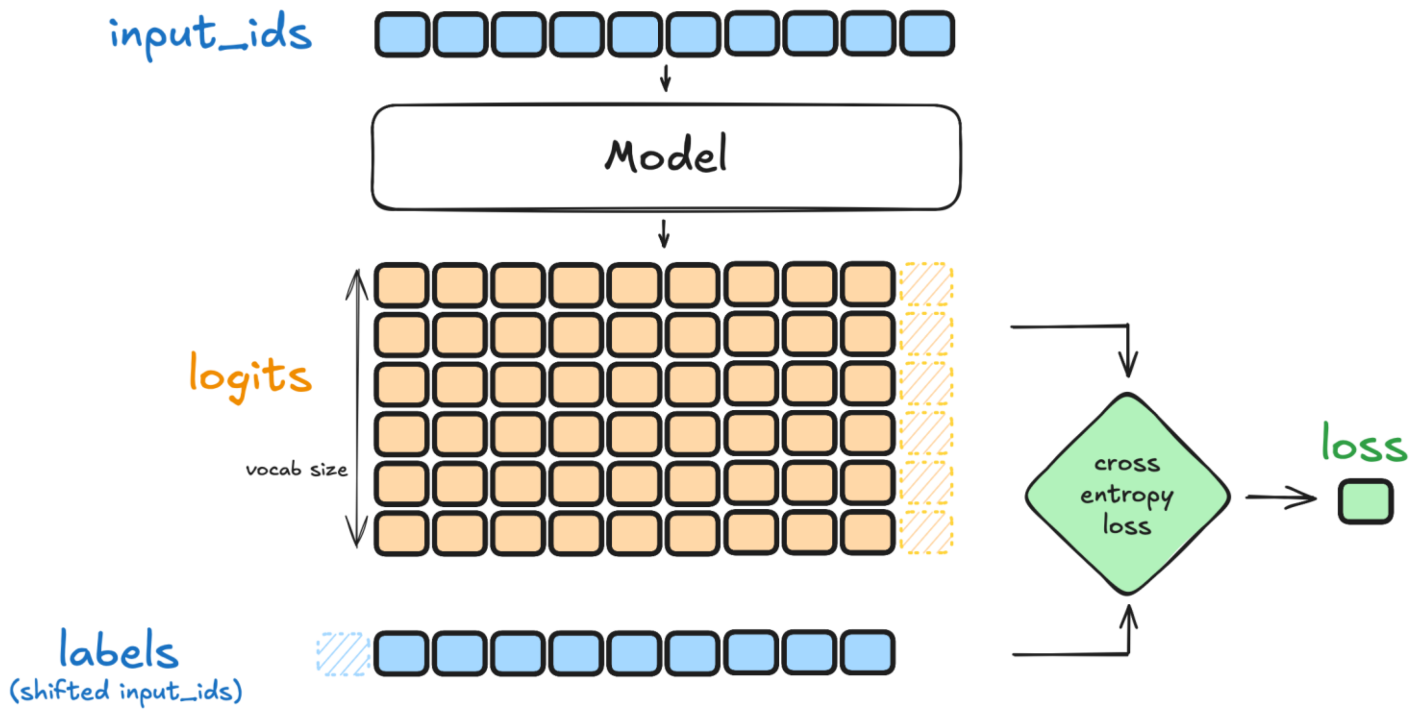
- Dataset Preparation and Selection: Quality and Relevance
  - Input-output pairs that show the behavior you want your model to learn.
  - Depending on model size: 1,000-10,000+ pairs
- Environment Setup and Configuration
- Training Configuration
  - Learning rate (5e-5 to 1e-4): control how much the model weights change with each update
  - Batch size: number of examples proceed simultaneously
  - Max sequence length: maximum tokens per training example

Fine-tuning Data Example



```
{
    "prompt": [
        {
            "content": "Given the symptoms of sudden weakness
            in the left arm and leg, recent long-distance
            travel, and the presence of swollen and tender
            right lower leg, what specific cardiac abnormality
            is most likely to be found upon further evaluation
            that could explain these findings?",
            "role": "user",
        }
    ],
    "completion": [
        {
            "content": "<think>Okay, let's see what's going on
            here. We've got sudden weakness [...] clicks into
            place!</think>The specific cardiac abnormality
            most likely to be found in [...] the presence of a
            PFO facilitating a paradoxical embolism.",
            "role": "assistant",
        }
    ],
}
```

Optimization Goal (Loss) in SFT

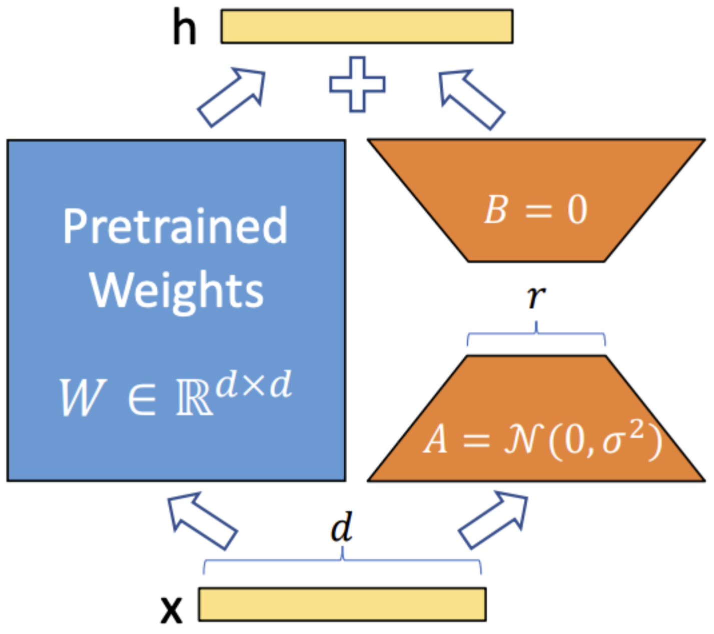


$$\mathcal{L}_{\text{SFT}}(\theta) = -\sum_{t=1}^{T} \log p_\theta(y_t \mid y_{<t})$$

where y_t is the target token at timestep t, and the model is trained to predict the next token given the previous ones.

Parameter-Efficient Fine-Tuning

- LoRA (Low-Rank Adaptation)
  - Freezes the pre-trained model weights
  - Decomposing the weight updates into smaller matrices through **low-rank decomposition**
  - Injecting trainable rank decomposition matrices into the transformer layers
  - LoRA reduced trainable parameters by 10,000x and GPU memory requirements by 3x compared to full fine-tuning when applied to GPT-3 175B
  - During inference, these adapter weights can be merged with the base model, resulting in no additional latency overhead


h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

LoRA: https://arxiv.org/pdf/2106.09685
OLoRA (QR Decomposition): https://arxiv.org/abs/2406.01775

Instruction Tuning VS. Preference Alignment

- **Instruction tuning** teaches models to follow instructions and engage in conversations;
- **Preference alignment** teaches models to generate responses that match human preference.
  - improved behavior across multiple areas
  - fewer harmful, biased, or inappropriate responses

Two main stages:

  - SFT: Adapt the base model to follow instructions through supervised fine-tuning.
  - DPO: Directly optimize the model using preference data through Direct Preference Optimization.

Reinforcement Learning from Human Feedback

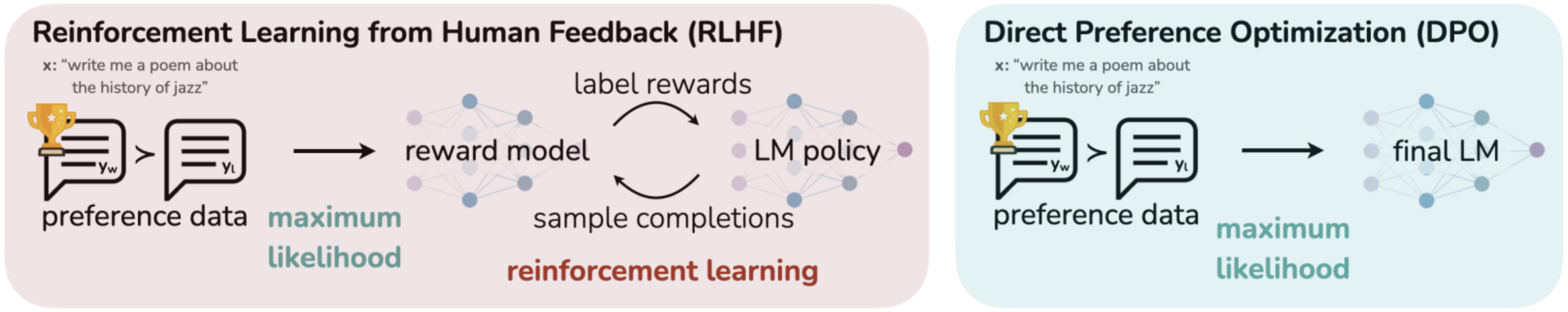

Reinforcement Learning from Human Feedback (RLHF)
x: "write me a poem about the history of jazz"
y_w
y_l
preference data
maximum likelihood
reward model
label rewards
sample completions
LM policy
reinforcement learning
Direct Preference Optimization (DPO)
x: "write me a poem about the history of jazz"
y_w
y_l
preference data
maximum likelihood
final LM

Figure 1: **DPO optimizes for human preferences while avoiding reinforcement learning.** Existing methods for fine-tuning language models with human feedback first fit a reward model to a dataset of prompts and human preferences over pairs of responses, and then use RL to find a policy that maximizes the learned reward. In contrast, DPO directly optimizes for the policy best satisfying the preferences with a simple classification objective, fitting an *implicit* reward model whose corresponding optimal policy can be extracted in closed form.

1. Training a reward model to predict human preferences based on preferred and rejected responses.
2. Using reinforcement learning algorithms like Proximal Policy Optimization (PPO) to optimize the policy against the reward model.

Direct Preference Optimization (DPO)

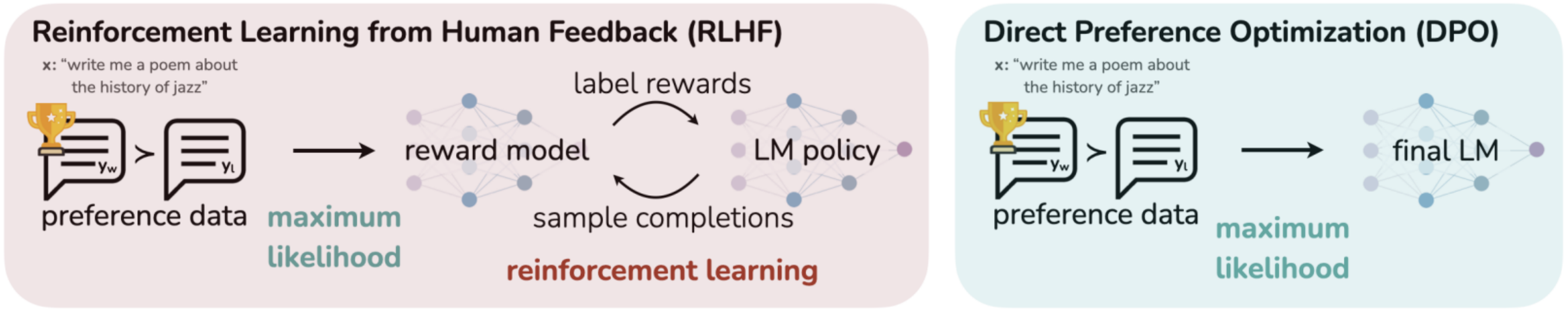

Reinforcement Learning from Human Feedback (RLHF)
x: "write me a poem about the history of jazz"
y_w
y_l
preference data
maximum likelihood
reward model
label rewards
sample completions
LM policy
reinforcement learning
Direct Preference Optimization (DPO)
x: "write me a poem about the history of jazz"
y_w
y_l
preference data
maximum likelihood
final LM

Figure 1: **DPO optimizes for human preferences while avoiding reinforcement learning.** Existing methods for fine-tuning language models with human feedback first fit a reward model to a dataset of prompts and human preferences over pairs of responses, and then use RL to find a policy that maximizes the learned reward. In contrast, DPO directly optimizes for the policy best satisfying the preferences with a simple classification objective, fitting an *implicit* reward model whose corresponding optimal policy can be extracted in closed form.

- DPO skips the reward model and using a binary cross-entropy loss to directly optimize the language model.
- DPO recasts preference alignment as a classification problem.

How DPO Works

- Given a prompt and two responses (one preferred, one rejected), DPO trains the model to increase the likelihood of the preferred response while decreasing the likelihood of the rejected response.
- The DPO process **requires supervised fine-tuning (SFT) first** to adapt the model to the target domain. This creates a foundation for preference learning by training on standard instruction-following datasets.
- Then, DPO implement binary classification to update model weights based on preference data.

DPO Loss Function

$$L_{DPO} = -\mathbb{E}_{(\pi,r)\sim D}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{ref}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)}\right)\right]$$

Where:

- `π_θ` is the model being trained
- `π_ref` is the reference model (usually the SFT model)
- `y_w` is the preferred (winning) response
- `y_l` is the rejected (losing) response
- `β` is a temperature parameter controlling optimization strength
- `σ` is the sigmoid function

DPO Dataset Sample

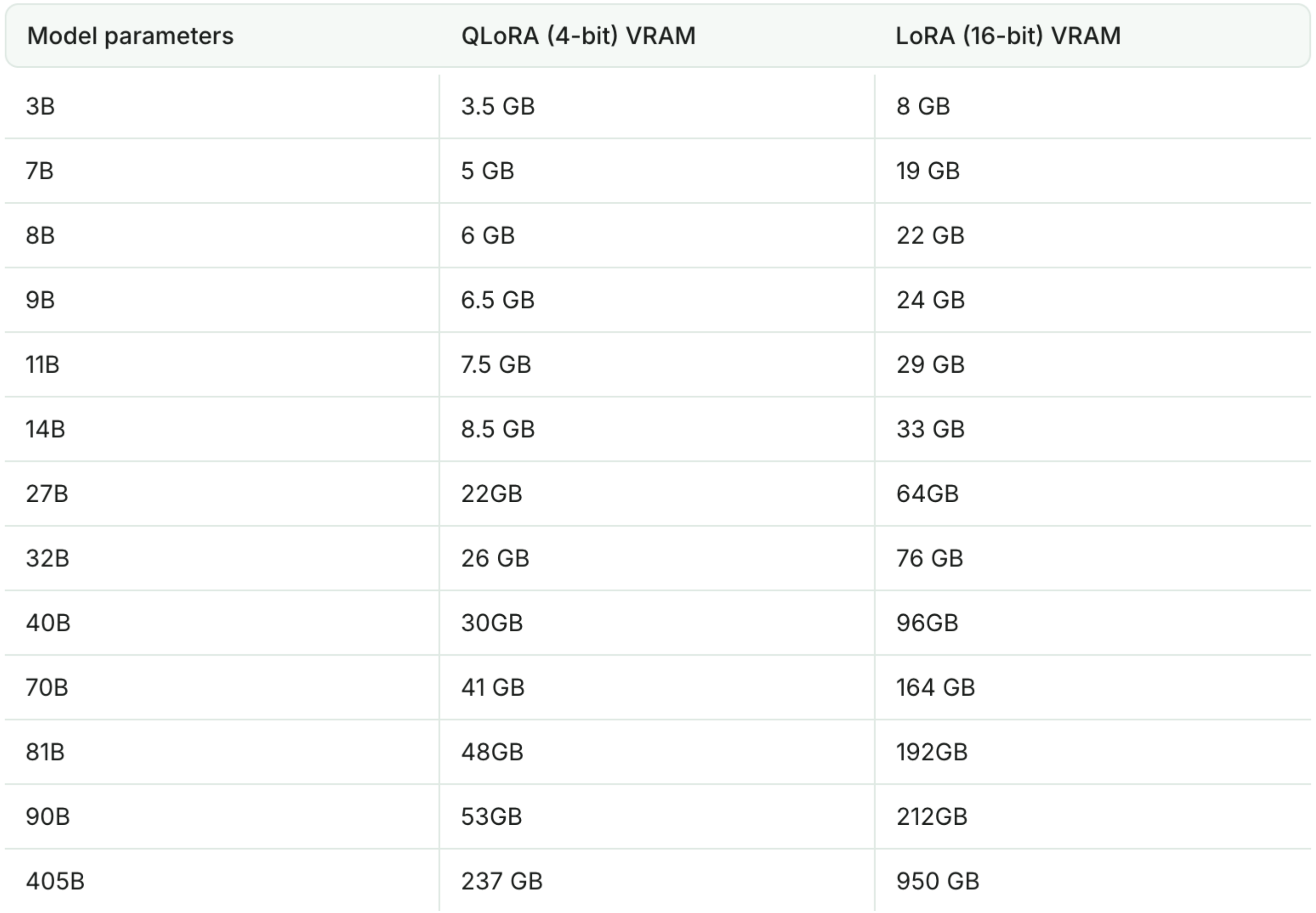
Field	Description	Example
`prompt`	The input prompt or question	“Explain quantum computing in simple terms”
`chosen`	The preferred response	“Quantum computing uses quantum mechanics principles…”
`rejected`	The less preferred response	“Quantum computing is very complex and hard to understand…”

Data Quality

- Data quality is crucial for successful DPO training.
- The preference dataset should include diverse examples covering different aspects of desired behavior.
- Clear annotation guidelines ensure consistent labeling of preferred and rejected responses.

- Compare the model’s outputs with the reference model (SFT’ed Model) to verify improvement in preference alignment.
- Check edge cases!

Next Lecture: Hands-on via Google Colab



Model parameters	QLoRA (4-bit) VRAM	LoRA (16-bit) VRAM
3B	3.5 GB	8 GB
7B	5 GB	19 GB
8B	6 GB	22 GB
9B	6.5 GB	24 GB
11B	7.5 GB	29 GB
14B	8.5 GB	33 GB
27B	22GB	64GB
32B	26 GB	76 GB
40B	30GB	96GB
70B	41 GB	164 GB
81B	48GB	192GB
90B	53GB	212GB
405B	237 GB	950 GB

https://unsloth.ai/docs/get-started/fine-tuning-for-beginners/unsloth-requirements

References

- https://huggingface.co/learn/llm-course/chapter2/1
- https://huggingface.co/learn/smol-course/unit1/3
- https://huggingface.co/learn/smol-course/unit2/2