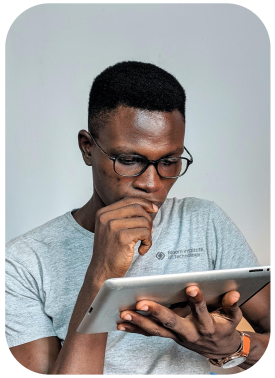
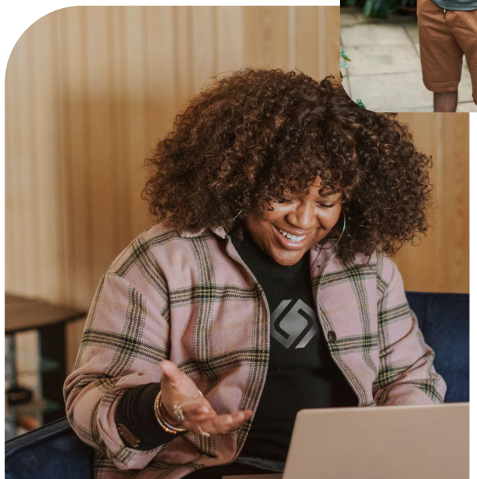
Bloom Institute of Technology

Advanced Fine-tuning Techniques

From PEFT/LoRA/QLoRA and Quantization to Human Feedback and Evaluation Methods.





Core Competencies

The student must demonstrate...

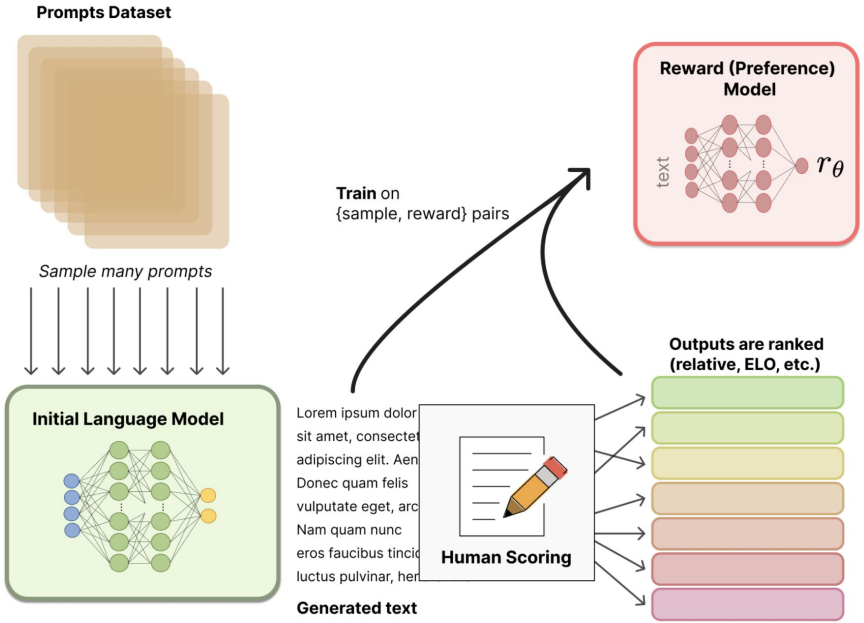
1. Understanding RLHF and how to use human feedback to train models. (10 min)
2. Proficiency in PEFT for updating a small set of model parameters efficiently. (10 min)
3. Understanding in LoRA to adapt models using low-rank matrices. (10 min)
4. Knowledge of Quantization and its role in reducing model size and cost. (10 min)
5. Understanding of QLoRA, combining quantization and LoRA for efficient model adaptation. (5 min)

Reinforcement Learning with Human Feedback

Reinforcement Learning with Human Feedback (RLHF) incorporates human input to improve AI decision-making. This ensures models are optimized to align with human preferences, resulting in more reliable AI.

How RLHF Works: Step-by-Step

1. **Pretraining the Language Model**: Begin with a pre-trained LLM like OpenAI's GPT-4o.
2. **Collect Human Feedback**: Have human evaluators review and rank model outputs based on their quality and alignment with desired outcomes.
3. **Train the Reward Model**: Use the collected human feedback to train a reward model that can score new outputs based on their alignment with human preferences.
4. **Fine-Tune with Reinforcement Learning**: Apply reinforcement learning, using the reward model to guide improvements.



Gather Feedback Using LangSmith

The code shows how to implement a simple feedback mechanism using the LangSmith client to record and score human feedback on the model's responses. It includes assigning a score and recording this feedback for further analysis and model improvement.

Score Prompts:

```
import os
from langsmith import Client

# Initialize the LangSmith client
client = Client()

def invoke_chatopenai(prompt):
   # Add logic to query OpenAI-3.5-turbo

# Define the scoring function
def score_prompt(run_id, feedback):
   score=1 if feedback=="Thumbs Up" else 0

   feedback_record = client.create_
   feedback(run_id, f"Feedback{score}")

   return score
```

Example Usage:

```
prompt = "Translate this sentence
from English to French. I love
programming."

run_id = "example_run_id"
feedback = "Thumbs Up"

response = invoke_chatopenai(prompt)
score = score_prompt(run_id, feedback)

print(f"Response: {response}")
print(f"Score: {score}")
```

PEFT

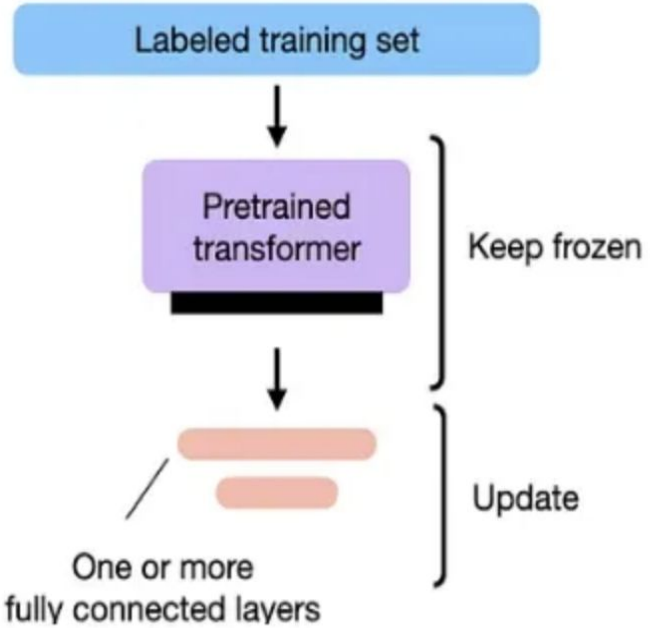
Parameter-Efficient Fine-Tuning (PEFT) is a method that fine-tunes a small subset of model parameters while keeping most of the pretrained model parameters frozen. This significantly reduces computational and storage costs, enabling the fine-tuning of large language models on consumer hardware.

Key Takeaways

- **Efficient Fine-Tuning**: PEFT updates a small subset of model parameters, reducing compute and storage requirements.
- **Cost-Effective**: Allows fine-tuning on consumer hardware, making advanced model training accessible.
- **Performance Maintenance**: Maintains the original model's performance while adapting to new tasks efficiently.

Considerations

- **Parameter Selection**: Careful selection of parameters to tune is crucial for optimal results.
- **Resource Management**: Balance computational savings with task-specific performance.

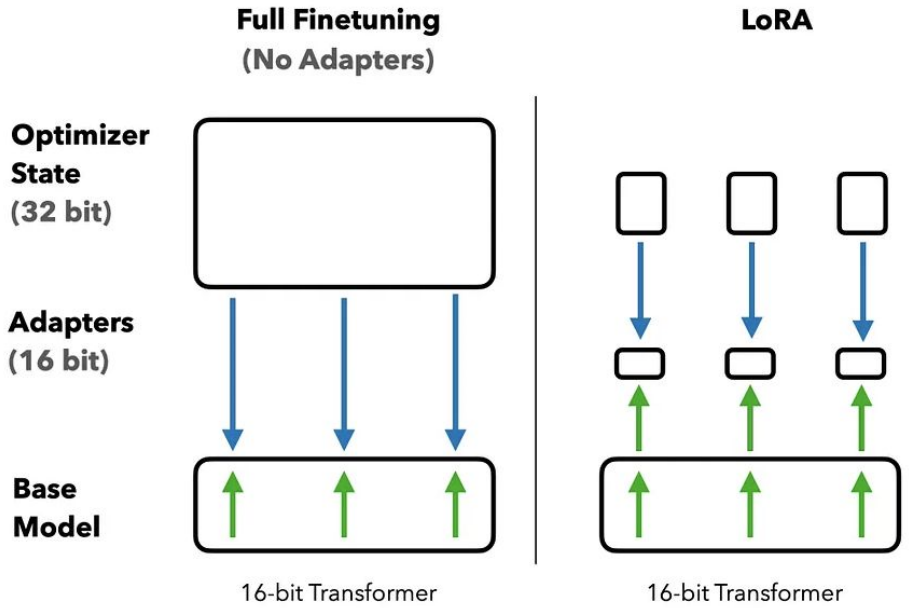


LoRA

Low-Rank Adaptation (LoRA) fine-tuning involves adding trainable low-rank matrices to the weights of a pre-trained LLM. These matrices are significantly smaller than the original weights, allowing only a fraction of the model's parameters to be updated during training.

How LoRA Works (Layman's Terms)

1. Start with a Pretrained Model: Begin with an AI model.
2. Add Extra Layers: Attach small, new layers (low-rank matrices) to the existing model.
3. Train the New Layers: Adjust these new layers while keeping the main model mostly the same. This is like tuning a few knobs without overhauling the machine.
4. Adapt to New Tasks: These small adjustments help the model learn new tasks quickly and efficiently.



Accessed 28 May 2024, <https://cameronrwolfe.substack.com/p/easily-train-a-specialized-llm-peft>.

Quantization

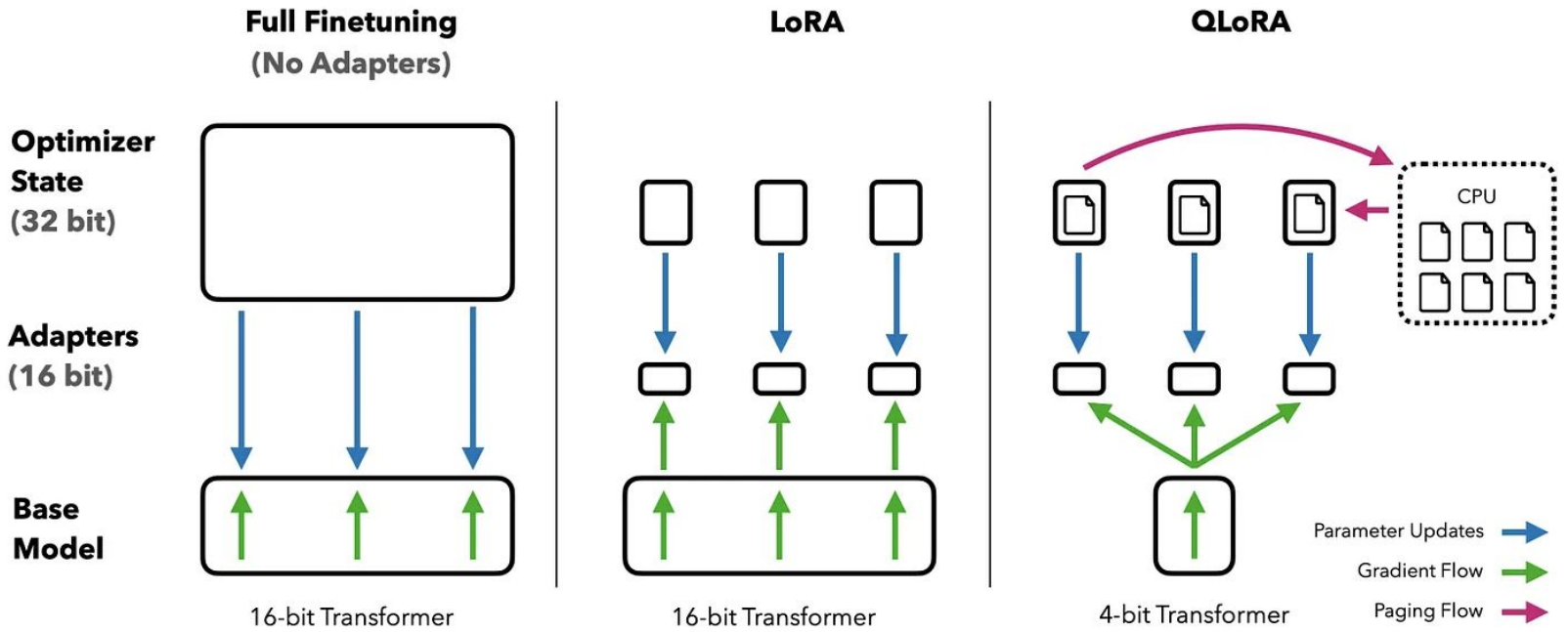
This process reduces the precision of a model's weights, typically from 32-bit floats to 8-bit integers, to make large language models more memory-efficient and faster. This technique enables the deployment of these models on less powerful hardware while maintaining performance.

Quantization and Its Use in QLoRA

- Reduced Precision: Quantization in QLoRA converts weights and activations to lower precision (e.g., 8-bit integers), significantly reducing memory usage.
- Efficient Fine-Tuning: By combining quantization with LoRA, QLoRA fine-tunes models using low-rank matrices, optimizing resource efficiency.
- Performance Retention: QLoRA maintains high performance by fine-tuning only the low-rank matrices while the quantized base model remains effective.
- Deployment on Limited Hardware: QLoRA enables the deployment of fine-tuned models on hardware with lower computational power, such as mobile devices and edge servers.

QLoRA

Quantized Low-Rank Adaptation (QLoRA) combines quantization with Low-Rank Adaptation (LoRA) to efficiently fine-tune large language models by converting weights and activations to lower precision.



Accessed 28 May 2024, <https://cameronrwolfe.substack.com/p/easily-train-a-specialized-llm-peft>.

PEFT vs. LoRA vs. QLoRA

	PEFT (Parameter-Efficient Fine-Tuning)	LoRa (Low-Rank Adaptation)	QLoRA (Quantized Low-Rank Adaptation)
Description	Updates a small subset of model parameters to reduce costs.	Injects low-rank matrices into model layers to adapt new tasks.	Combines quantization with low-rank adaptation for tuning.
Method	Selectively fine-tunes specific parameters.	Adds low-rank matrices to the model's architecture.	Applies quantization to weights and activations, then uses LoRA.
Benefits	Less resource-intensive, faster fine-tuning process.	Maintains performance with minimal additional parameters.	Lowers compute requirements while preserving performance.
Use Cases	Scenarios requiring efficient fine-tuning with limited resources.	Tasks needing quick adaptation of large models without retraining.	Tuning LLMs on limited hardware while ensuring efficiency.
Challenges	May require careful selection of parameters to tune.	Complexity in integrating low-rank matrices.	Balancing quantization precision and adaptation effectiveness.

Fine-Tuning Llama2 with PEFT and QLoRA in Colab

Let's walk through fine-tuning Llama2 using the techniques we learned in class. Make sure your Colab runtime is set to `T4 GPU`. Access the complete code here.

Prerequisites Before You Explore Code

1. **Understand Hyperparameters**: Hyperparameters are the "magic constants" that must be set before training or fine-tuning a model. Unlike regular parameters, hyperparameters are chosen by humans.
2. Key Hyperparameters to Consider:
    a. **Learning Rate**: Determines how quickly the model adjusts its parameters during training.
    b. **Dropout**: Helps prevent overfitting by randomly setting a fraction of input units to zero during training.
    c. **Alpha**: A parameter that can vary in definition depending on the context but is crucial in some models.
3. Monitoring and Validation
    a. **Tracking Progress**: Tools like TensorBoard help monitor the loss over time. As long as the loss is decreasing, it's reasonable to continue training.
    b. **Validation**: Hyperparameter tuning underscores the importance of using holdout data to validate the model. This prevents overfitting to the training data and ensures better generalization.

Hands-on Homework.

Using an existing dataset on `HuggingFace`, fine-tune an open-source LLM, like `llama2`, using PEFT and QLoRA. Reference this code, when fine-tuning your own model.

Remember to change the `runtime` on Colab to `T4 GPU`.