Fine-tuning an LLM may not be as trivial as we may think! Depending on your data, it may lead to the model forgetting what it learned in the pretraining phase! You want to fine-tune it, but you also may want to retain its coding or chatting abilities. Because you most likely don't have the right benchmark data to validate it on different learning tasks, it might be difficult to understand the abilities it lost in the process!   
  
Why would we want to fine-tune an LLM in the first place? There 2 main reasons! First, we may want to augment the model's data bank with private data, and second, we may want the model to specialize in specific learning tasks. A full fine-tuning takes time and money and generates a very large resulting model file. The typical way to go about it is to use Low-Rank Adaptaters (LoRA) to minimize the fine-tuning cost.   
  
The idea is to replace within the model some of the large matrices with smaller ones for the gradient computation. Let's call W0 the weights of the pre-trained model for a specific layer matrix. After a gradient update ΔW, the weights will be  
  
W = W0 + ΔW  
  
and, if x is the input to that layer, the output of that layer will be  
  
W . x = W0 . x + ΔW . x  
  
If we use the LLama2 with 70B parameters, we need to update all the parameters for each backward pass: computationally very expensive! Instead, with LoRA, we insert next to each layer matrix of the pre-trained model, 2 matrices A and B, such that the update is approximated by a lower rank decomposition:  
ΔW ~ B . A   
  
The trick is that if ΔW has dimensions (R, C), we can create B with dimensions (R, r) and A with dimensions (r, C) such that r << R, C. For example if R = 10K, C = 20K and r = 4, then   
  
ΔW has R x C = 10K x 20K = 200M elements  
B has R x r = 10K x 4 = 40K elements  
and A has r x C= 20K x 4 = 80K elements  
  
Therefore A and B combined have 120K elements which is 1666 times less elements than ΔW. When we fine-tune, we only update the weights of those newly inserted matrices. The gradient matrices are much smaller and, therefore, require much less GPU memory space. Because the pre-trained weights are frozen, we don't need to compute the gradients for a vast majority of the parameters.   
  
To gain even more space, we may want to quantize the float parameters into integers while applying LoRA (QLoRA). Now, the number of fine-tuned weights is just a fraction of the original model size, and we can more easily store those weights for each of the learning tasks for which we needed fine-tuning. When we need to deploy an inference server, we can use the original pre-trained model and combine it with the fine-tuned LoRA adapters for the specific learning task needed on that server.

