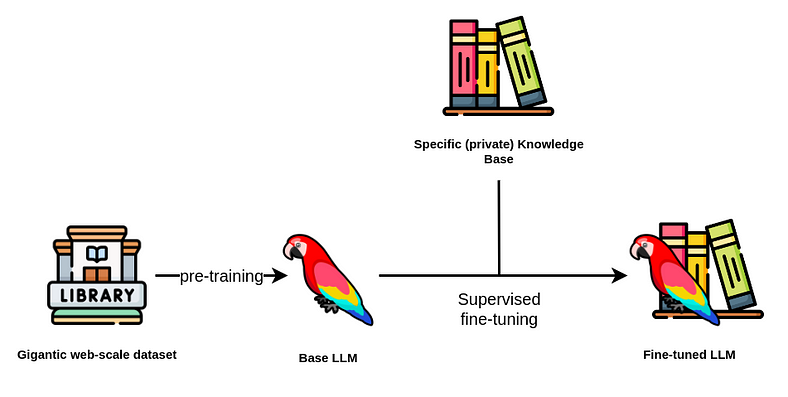
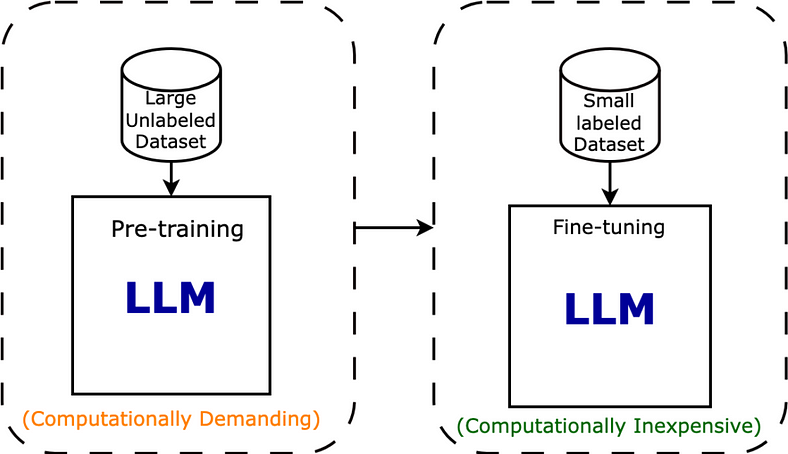
Supervised Fine-tuning: customizing LLMs



Supervised Finetuning on LLMs. Source: [Neo4j](https://neo4j.com/developer-blog/fine-tuning-retrieval-augmented-generation/)

In the rapidly evolving field of Natural Language Processing (NLP), fine-tuning has emerged as a powerful and effective technique to adapt pre-trained Large Language Models (LLMs) to specific downstream tasks. Pre-trained large-scale language models (as GPT family) have shown significant advancements in language understanding and generation. However, these pre-trained models are typically trained on vast amounts of text data with unsupervised learning and may not be optimized for a specific task.

Fine-tuning bridges this gap by taking advantage of the general language understanding captured during pre-training and adapting it to a target task through supervised learning. By fine-tuning a pre-trained model on a task-specific dataset, NLP practitioners can achieve impressive results with significantly less training data and computational resources than training a model from scratch. Specifically, for Large Language Models, finetuning is crucial, as a retraining step with the whole data is computationally prohibitive.



(Pre)training an LLM vs Fine-tuning. Source: [Intuitive Tutorials](https://intuitivetutorial.com/2023/06/18/large-language-models-in-deep-learning/)

The success of fine-tuning has led to numerous state-of-the-art results across a wide range of NLP tasks and has become a standard practice in the development of high-performing language models. Researchers and practitioners alike continue to explore variations and optimizations of fine-tuning techniques to push the boundaries of NLP capabilities further.

In this article, we will delve deeper into the process of fine-tuning an Instruction-based Large Language Model using transformers library in two different ways: with just the basic transformers library and with the trl module.

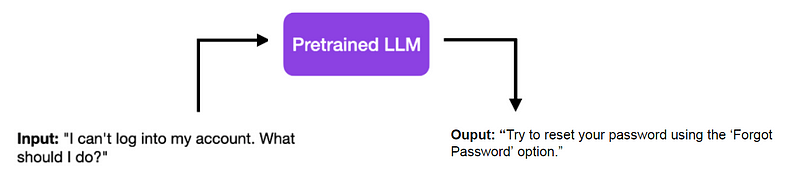
Supervised fine-tuning (SFT)

Supervised fine-tuning, involves adapting a pre-trained Language Model (LLM) to a specific downstream task using labeled data. In supervised fine-tuning, the finetuning data is collected from a set of responses validated before hand. That’s the main difference to the unsupervised techniques, where data is not validated beforehand. While LLM training is (usually) unsupervised, Finetuning is (usually) supervised.

During supervised fine-tuning, the pre-trained LLM is fine-tuned on this labeled dataset using supervised learning techniques. The model’s weights are adjusted based on the gradients derived from the task-specific loss, which measures the difference between the LLM’s predictions and the ground truth labels.

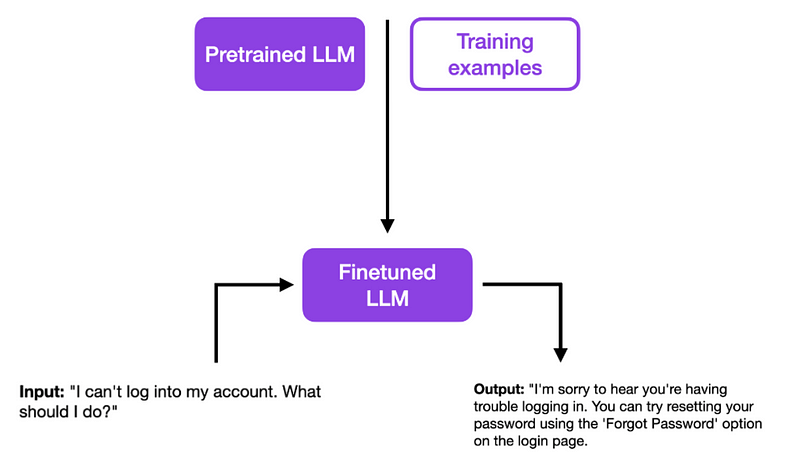
The supervised fine-tuning process allows the model to learn task-specific patterns and nuances present in the labeled data. By adapting its parameters according to the specific data distribution and task requirements, the model becomes specialized in performing well on the target task.

For example, let’s suppose you have a pretrained LLM. To the input I can't log into my account. What should I do? it answers with a simple Try to reser your password using the "Forgot Password" option.



A dry and concise answer to a Customer Support question, using Pretrained LLMs.

And now imagine you want to build a chatbot for a Customer Support service. Although the answer below may be correct, it is not adequate as it is as a Customer Support answer, which would require more empathy, a different format, additional contact details, or whatever your guidelines are. This is when Supervised Finetuning comes into play.



A better answer to a Customer Support question matching some guidelines, after Finetuning.

By providing a series of validated Training Examples, your model can learn to answer better to prompts are questions. In the example of the illustration below, we taught the model some [Customer Support empathy](https://acxpa.com.au/empathy-statements-for-customer-service/) statements.

Some of the reasons you may want to use finetuning of LLMs are:

Achieve better answers, matching your business *guidelines*, as explained above.

Provide new specific / private data, which were not available publicly during the training step, so that the LLM model is adapted to your specific *knowledge base*.

Teach the LLM to answer new (unseen) *prompts*;

Proximal Policy Optimization (PPO) is a type of reinforcement learning algorithm used in machine learning for training agents to interact with an environment and learn optimal behavior. It is specifically designed for problems where the agent makes sequential decisions over time, such as in robotics, game playing, and autonomous driving.

Here are some key characteristics and components of PPO:

1. \*\*Policy Gradient Methods:\*\* PPO belongs to the family of policy gradient methods, which directly parameterize the policy of the agent and optimize it to maximize cumulative rewards.

2. \*\*Stability and Sample Efficiency:\*\* PPO is developed to address some of the limitations of earlier policy gradient methods, such as instability in training and inefficiency in sample usage. It achieves better stability and sample efficiency by using a clipped surrogate objective function.

3. \*\*Clipped Surrogate Objective:\*\* PPO uses a clipped surrogate objective to update the policy parameters. Instead of directly optimizing the policy towards maximizing expected rewards, PPO constrains the policy update to be within a certain threshold. This prevents large policy updates that may destabilize training.

4. \*\*Trust Region Optimization:\*\* PPO employs a trust region optimization approach, which ensures that the updated policy does not deviate too far from the previous policy. This helps maintain stability during training and prevents catastrophic updates.

5. \*\*Actor-Critic Architecture:\*\* PPO often uses an actor-critic architecture, where two neural networks are employed: an actor network that learns the policy and a critic network that estimates the value function. This combination allows PPO to utilize both policy-based and value-based methods for more effective learning.

6. \*\*Batched Sampling:\*\* PPO typically samples multiple trajectories or batches of experiences from the environment to update the policy. This batched sampling approach improves sample efficiency and accelerates training.

Overall, Proximal Policy Optimization is a powerful and widely used reinforcement learning algorithm that has demonstrated success in a variety of challenging tasks, including complex games, robotic control, and autonomous navigation. Its stability, efficiency, and ease of implementation make it a popular choice for researchers and practitioners in the field of reinforcement learning.

In May 2023, this algorithm was introduced in [Direct Preference Optimization: Your Language Model is Secretly a Reward Model](https://arxiv.org/abs/2305.18290).

DPO comes as a direct alternative to [Reinforcement Learning from Human Feedback (RLHF)](https://pub.towardsai.net/reinforcement-learning-from-human-feedback-rlhf-f88687d5402e) and has been gaining a lot of popularity since **it does not require a reward model**. It was recently used in [Zephyr](https://arxiv.org/abs/2310.16944), the best 7B language model at the time of writing.

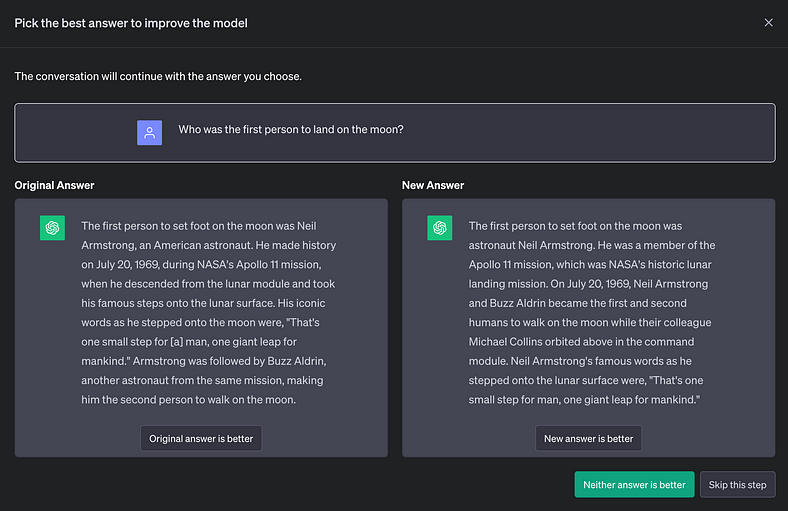
This method is applied to generative language models (LMs), such as GPT, Llama, Zephyr, and T5.

The objective of DPO is the same as RLHF: improve the alignment of language models to human preferences.

**Preference Data**

DPO is applied to preference data, which basically consists of a dataset of triplets (prompt, chosen answer, rejected answer).

In other words, for each prompt, there is a better response and a worse response.



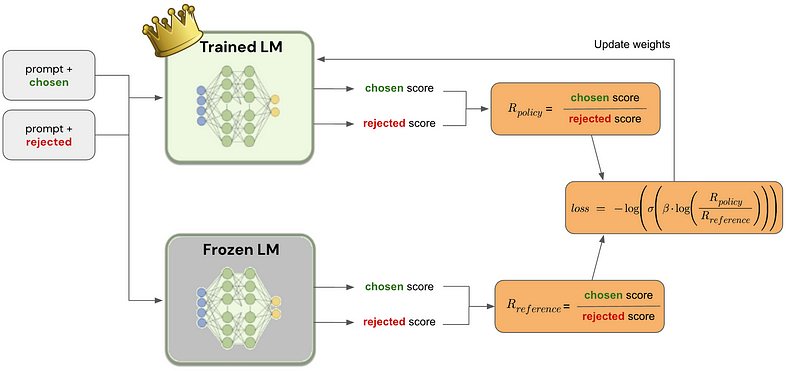
Preference data collection in ChatGPT (Image by author)

This type of data is also collected for RLHF, with the objective to train a reward model, that is later used to train the model with reinforcement learning.

**In DPO, there is no reinforcement learning, and the model is directly optimized in this preference data.**

Also interesting to note, unlike RLHF, these responses do not have to be sampled from the language model that we are optimizing.

**DPO Fine-tuning**

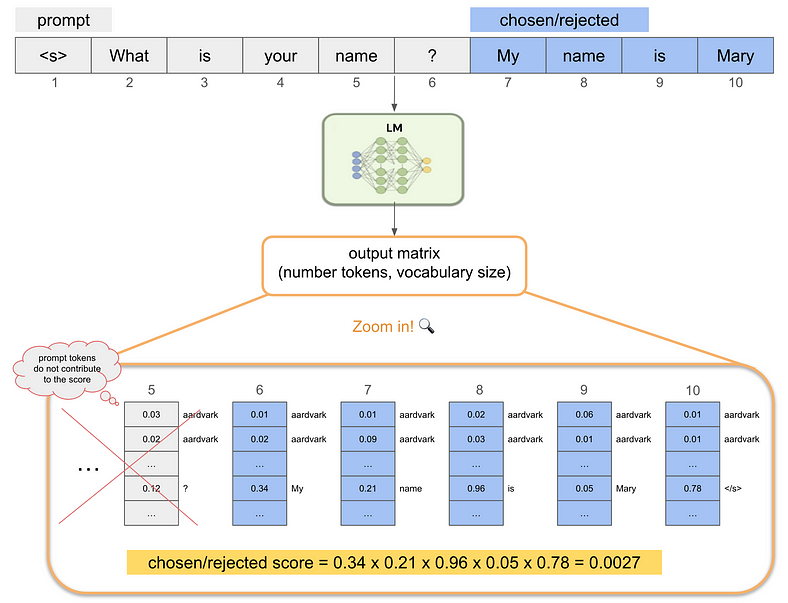


Fine-tuning the main language model (LM) using DPO (Image by author)

At the beginning of the fine-tuning process, an exact copy is done for the language model (LM) that is being trained, and its trainable parameters are frozen.

For each datapoint, the chosen and rejected responses are scored by both the trained and the frozen language model. This score is the product of the probabilities associated with the desired response token for each step.

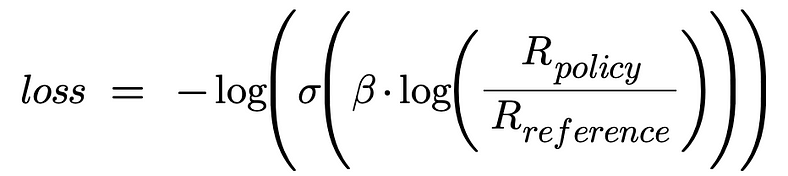
Since these generative language models use a causal decoder, we can calculate this score in a single forward pass, as seen in the image below.



How the language model (LM) scores a chosen/rejected response for a given prompt: for each generation step, the probability of generating the response token is selected, and these probabilities are multiplied at the end (Image by author)

With the chosen and rejected responses scored, we can calculate the ratio between the scores given by the trained language model, ***R****policy*, and the ones given by the frozen language model, ***R****reference*.

These ratios are then used to calculate the final loss that is used to modify the model weights in the gradient descent update:



DPO loss (Image by author)

where **β** is a hyperparameter ([Zephyr](https://arxiv.org/abs/2310.16944) used β=0.1) and **σ** is the sigmoid function.

*Note: you may see this loss equation written in log-scale, which is the recommended way to implement it, for numerical stability.*

**Conclusions**

Direct Preference Optimization is a stable, performant, and computationally lightweight algorithm.

Unlike its predecessor, RLHF, DPO eliminates the need for fitting a reward model, sampling from the language model during fine-tuning, or performing significant hyperparameter tuning.

In summary, DPO is not just a new algorithm; it’s a game-changer, simplifying and enhancing the way we build language models that truly understand and cater to human needs.