|  |  |
| --- | --- |
| GEN AI LEARNING  **Complete Generative AI Learning – New Year Challenge** | WRITTEN BY: ALOY |

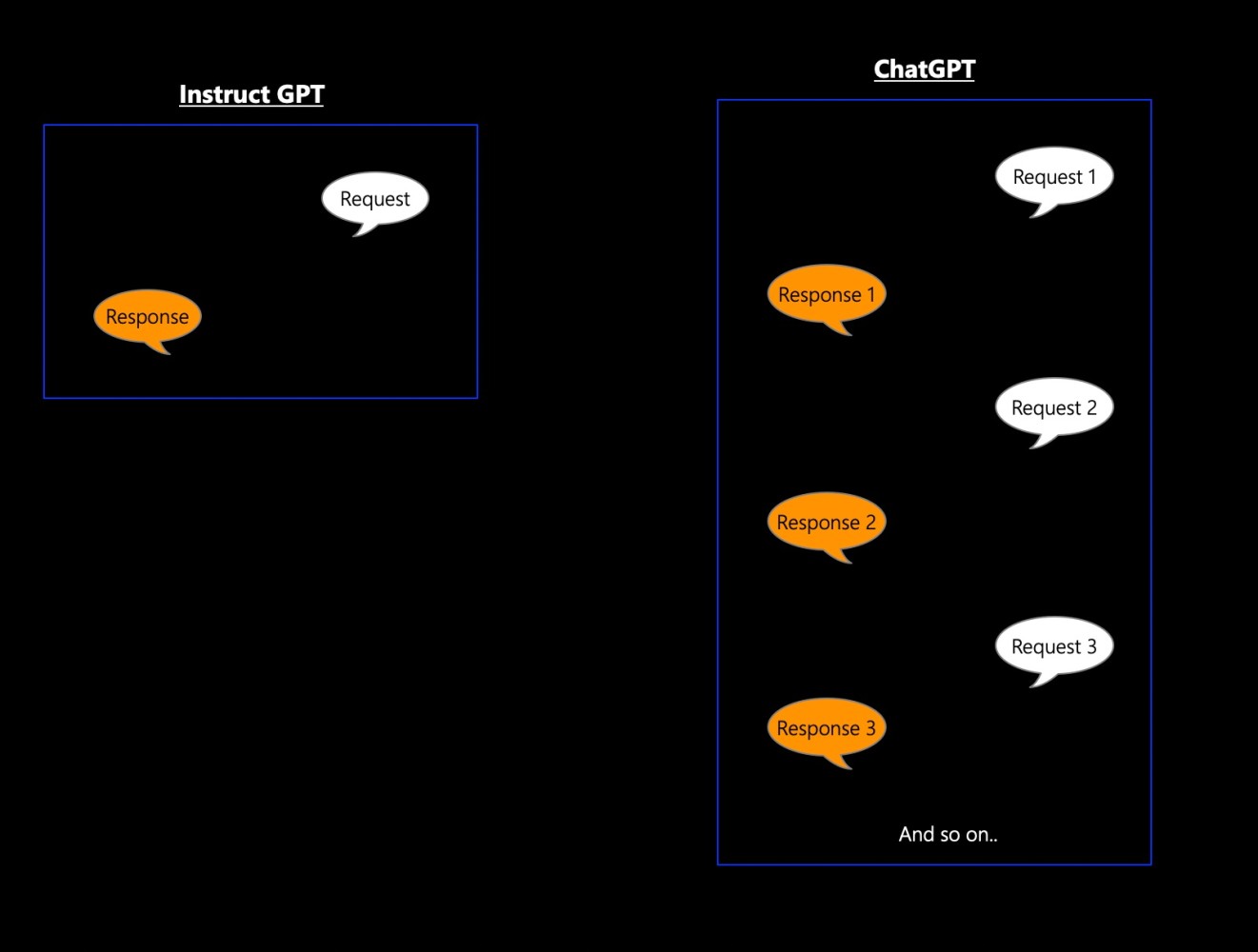
***Day – 14***

**Gen AI Notes:**

**Model Genesis**

If you’ve used ChatGPT before, you know what’s up. But, before we discuss about ChatGPT and how it gets trained, we have to discuss its predecessor.

To train ChatGPT, a similar method to [InstructGPT](https://arxiv.org/pdf/2203.02155.pdf) is used. However, there are some big differences between the two models. Check out this diagram to see how ChatGPT does it differently from InstructGPT.



InstructGPT was originally meant to be all about following instructions. You give it one request and it gives you one response. But ChatGPT takes that idea and kicks it up a notch. ChatGPT can handle multiple requests and responses while keeping the context of the conversation.

**Stages of Training ChatGPT**

To pull off this awesome trick, ChatGPT needs some serious training, broken down into three stages. Here’s a sweet diagram that gives you an overview on all the stages:



Let’s take a closer look at each of these stages.

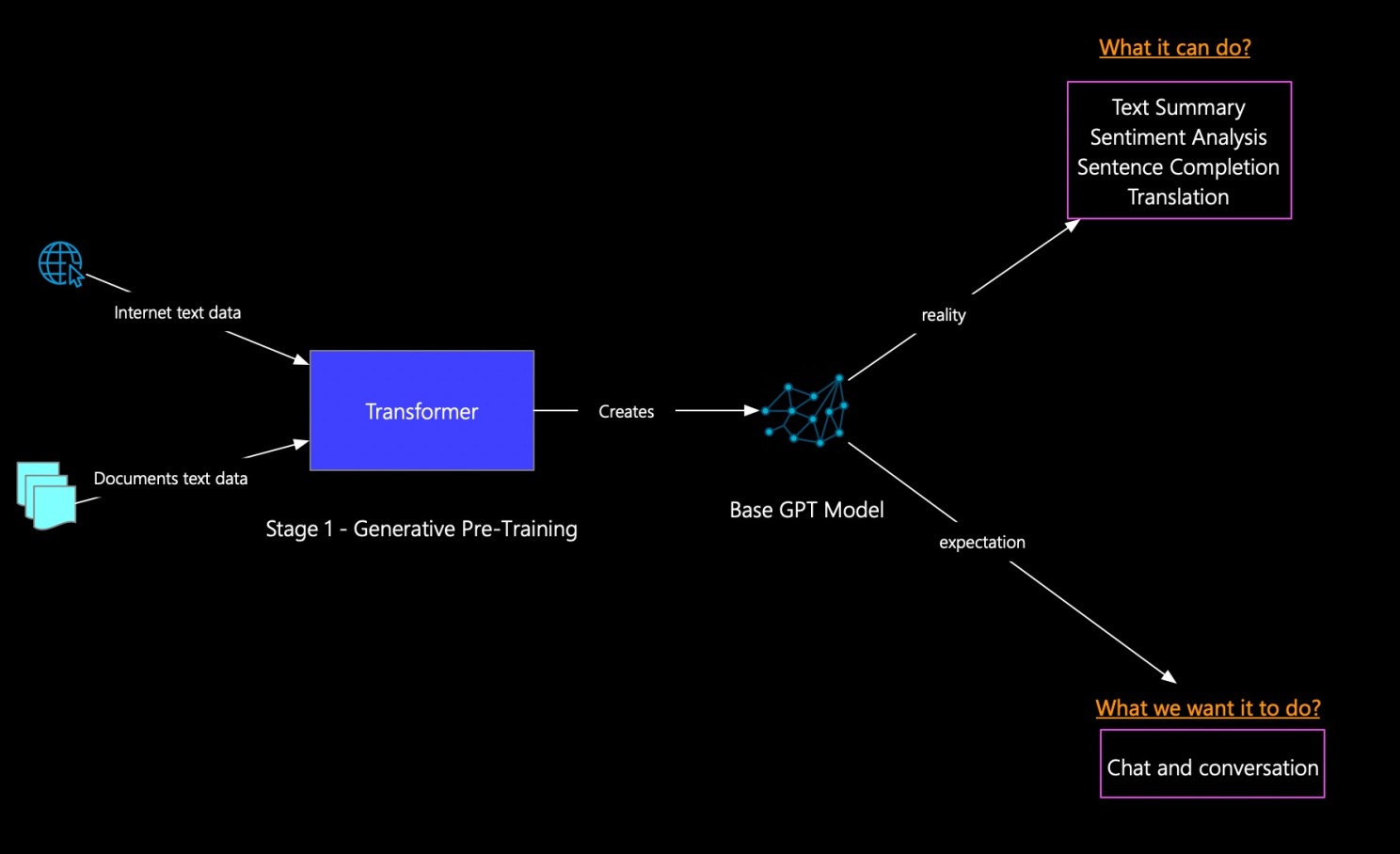
**Stage 1 — Generative Pre-Training**

In the first stage of training, the transformer is in full throttle. Basically, it is trained on a bunch of text data from all over the internet — websites, books, articles, you name it. It is a variety of genres and topics so it can really get the hang of generating text in different styles and contexts.

We went even deeper into the guts of the transformer in the blog post [Introduction to Large Language Models and Transformer Architecture](https://www.linkedin.com/pulse/introduction-large-language-models-transformer-pradeep-menon/). Check that out for more information.

It is important to understand why just doing this one thing isn’t going to cut it for ChatGPT to get the results it does.

Fundamentally, there is a misalignment in expectations here. The following diagram tries to explain why things aren’t quite lining up.



The users got some expectations about what ChatGPT can do, but it seems like they’re a bit out of sync with what the base GPT model is capable of. Stage 1 of the model is training to do a lot of things like language modeling, summarization, translation, and sentiment analysis. It’s not trained for a specific task but can handle a bunch of different ones.

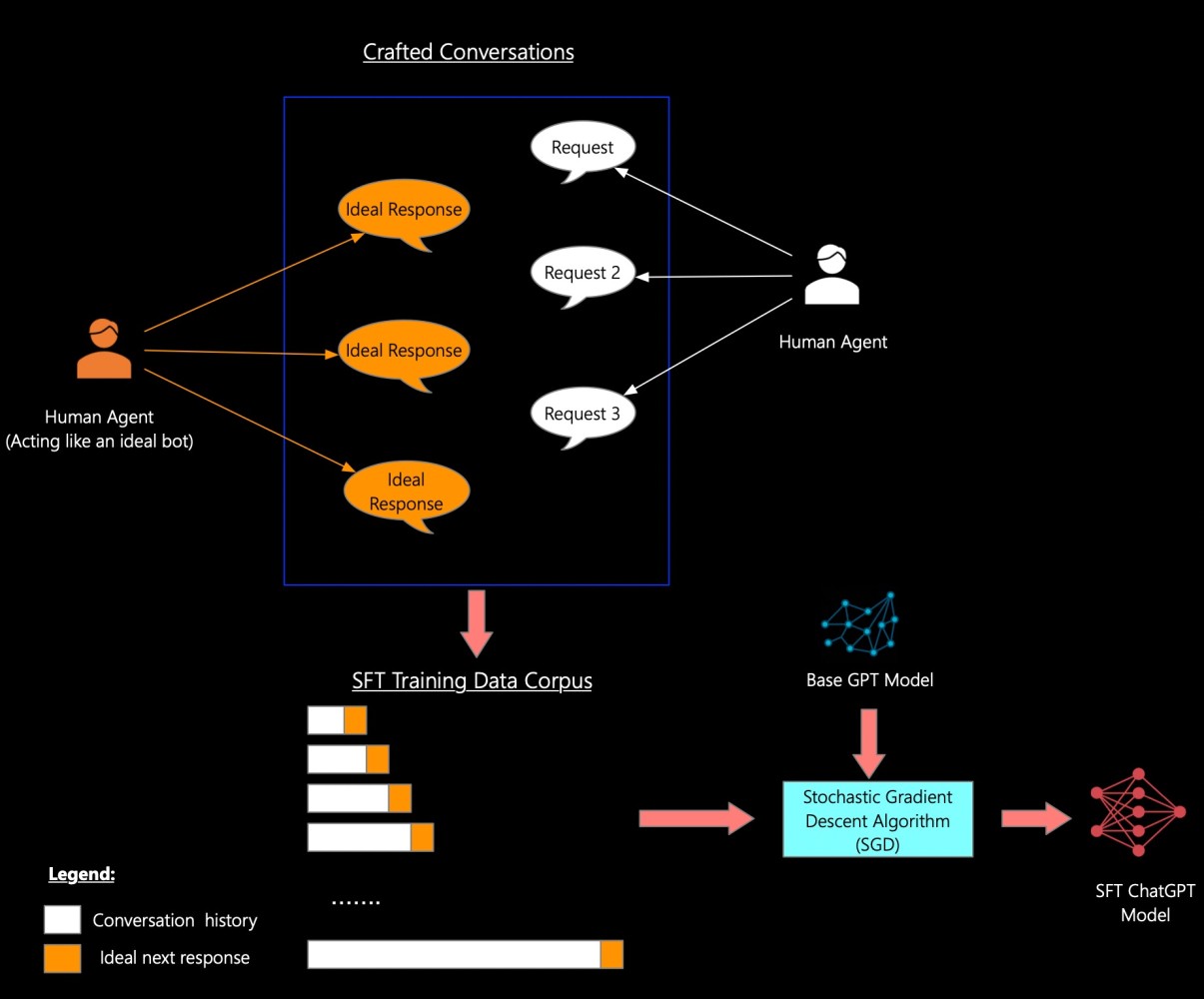
For example, it’s great at text completion, where it can generate the next word or sentence based on the context given in the prompt. It’s also really good at text summarization, where it can take a massive article and boil it down into something shorter.

But, the user seems to think that ChatGPT can chat about a particular topic. Unfortunately, that’s just not what the model is built to do. The expectation is misaligned with what the model is actually capable of doing.

Because of this misalignment, we got to fine-tune the base GPT model some more to make sure it meets the expectations. This brings us to the next part of our training stage: Supervised Fine-Tuning.

**Stage 2 — Supervised Fine-Tuning (SFT)**

Supervised Fine-Tuning (SFT) is the second round of training for ChatGPT. During this stage, the model gets trained on specific tasks that are relevant to what the user is looking for, like conversational chat. The idea is to make the model even better at meeting the user’s expectations and crushing it on the task. The following diagram shows how the base model is fine-tuned using SFT.

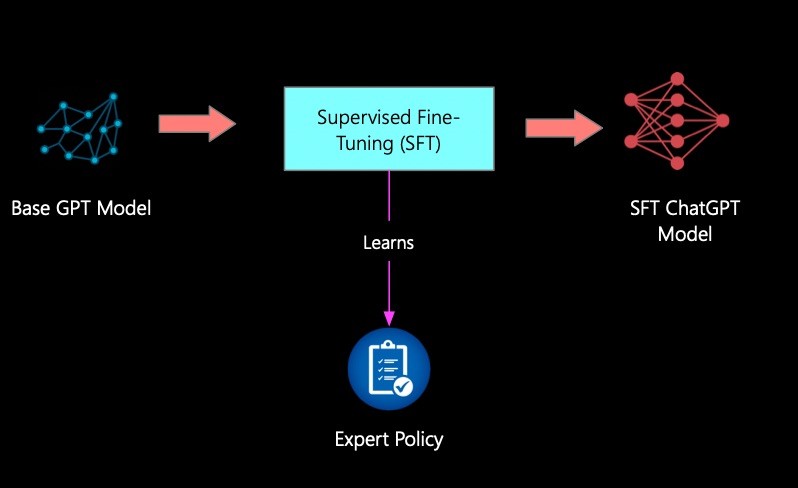


Let’s take a closer look and see what’s up in SFT. SFT is a three-step process:

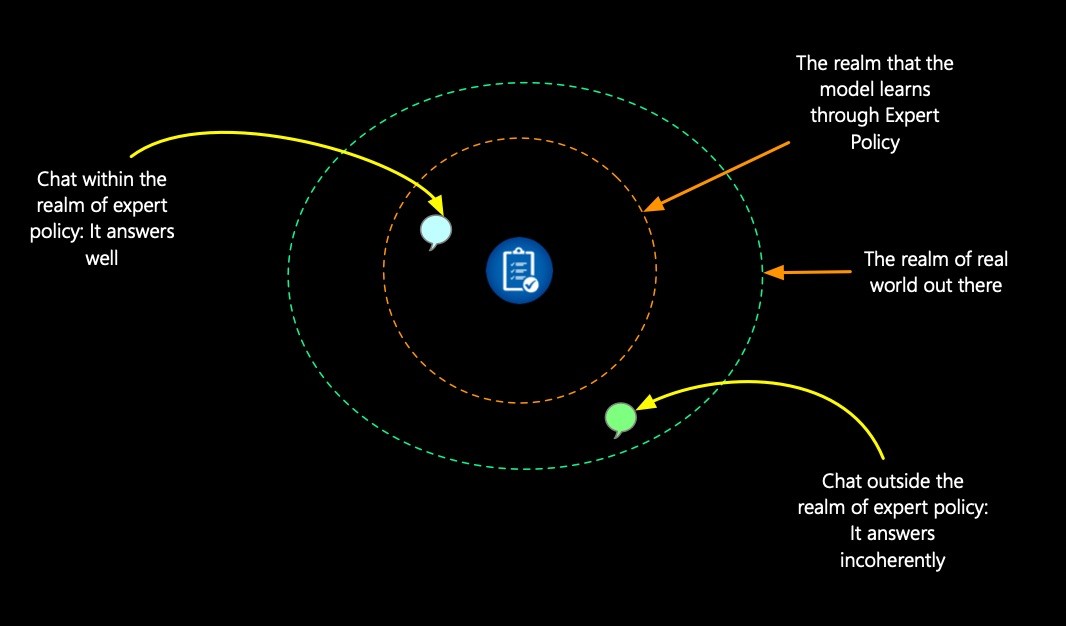
1. The first step during the SFT stage is to create these sets of carefully crafted conversations. These conversations are created by one human agent chatting with another human agent pretending to be a chatbot. The human pretending to be the chatbot will give the ideal response to each request. Then, tons of these conversations and used to create the training data corpus for SFT.
2. The next step is to create the training corpus, which involves using the conversation history as input and aligning it with the ideal next response as output. This creates a set of tokens on which the Base GPT model’s parameters are updated.
3. Alrighty, so the SFT training corpus is trained using Base GPT Model and this fancy thing called Stochastic Gradient Descent Algorithm (SGD). Think of Stochastic Gradient Descent (SGD) as a teacher that helps a computer learn how to win a game. First, the teacher shows the computer how to play and the rules. Then, the teacher watches and gives tips on how to improve. After each game, the teacher tells the computer what it did wrong and how to improve. This process repeats until the computer gets really good at the game. SGD is like this optimization algorithm for machine learning that keeps tweaking the parameters of a model until the cost function is minimized. At each step, the algorithm randomly picks a subset of the training data (aka mini-batch) to calculate the gradient of the cost function with respect to the parameters. Then, the parameters get updated based on the gradient computed from the mini-batch. This process happens again and again until the algorithm finally reaches the minimum point of the cost function.

During the Supervised Fine Tuning stage, the parameters of the ChatGPT base model are updated to capture task-specific info that wasn’t around before SFT.

We’re almost done with the training! However, before we dive into the final stretch, let’s discuss why the ChatGPT model still isn’t quite there even after all that SFT action. The issue that ChatGPT faces, even after SFT, is known as the “**Distributional Shift**.” Let’s try to understand it better using the following diagrams:



SFT uses this technique called “imitations”. Basically, they teach their model by having it mimic how humans respond in conversations. The model then creates an expert policy, which acts like a rule book for how the model should respond to requests. This policy is based on the conversations that SFT used to train the model. Check out this diagram that shows how the distributional shift happens.

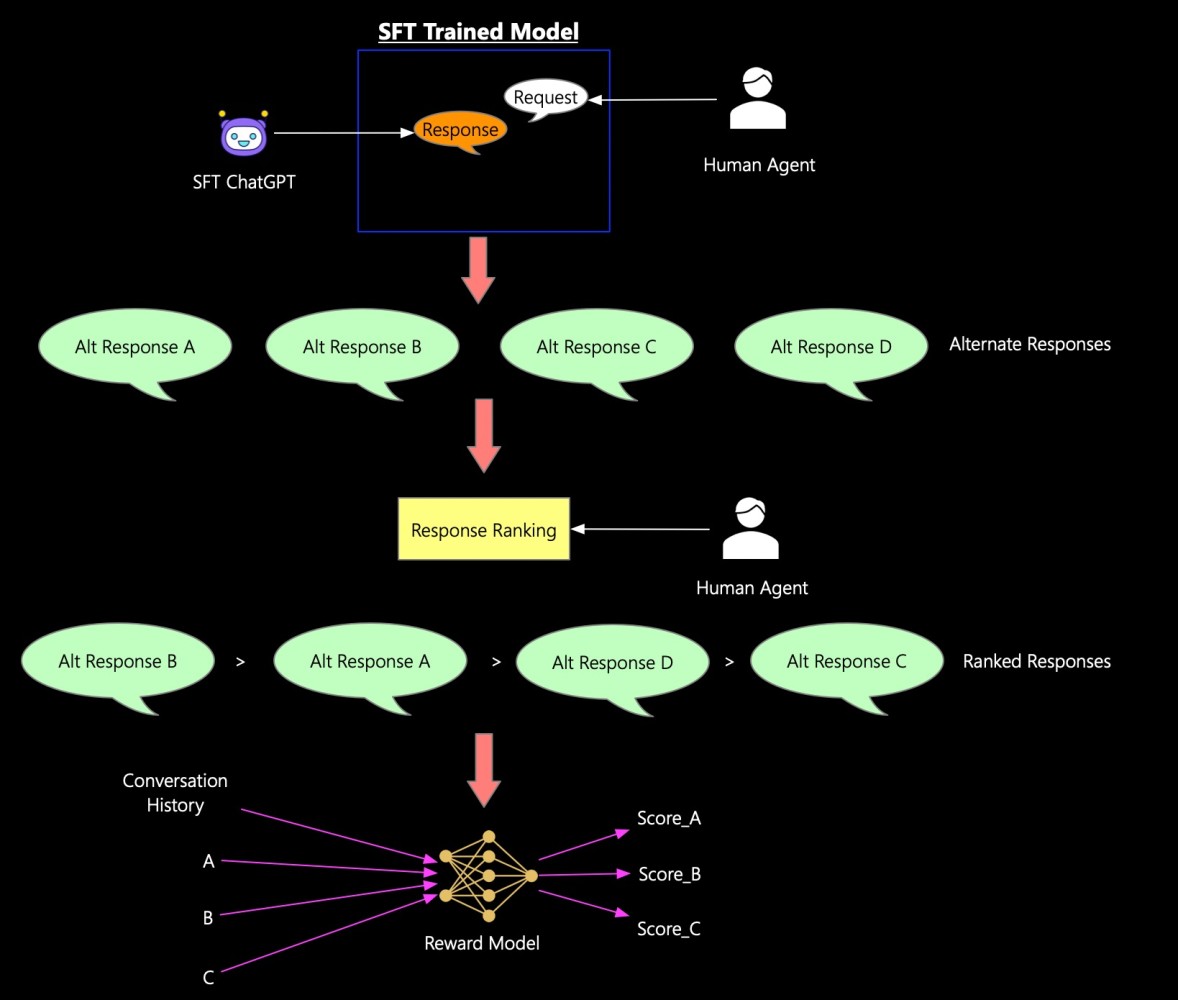


The idea is pretty straightforward. Even if you throw all kinds of chats and texts at this model, it’s not going to magically know everything. ChatGPT only knows what it has been taught. It’s like a tiny little piece of the world that was copied into its brain. So if you ask it something that’s not in that piece, it’s gonna freak out and give you some random answer.

To keep this drift in check, the model needs to act proactively during the conversation and not passively answer what it has learned. This learning is done in stage three with Reinforcement Learning through Human Feedback (RLHF). Let’s dive in and see how it works!

**Stage 3 — Reinforcement Learning through Human Feedback (RLHF)**

In Reinforcement Learning (RL) the agent interacts with its environment and learns to make decisions by getting rewarded or punished. It is like training a puppy, but with computers. The way success is measured in RL is through a “reward function.” It’s basically a way of turning our goals into a number that we can use to see how well the agent is performing. By focusing on getting a high score in this reward function, the agent can get better and better at making good decisions. When we are training the ChatGPT model in Stage three, we use a human agent as well to do the RL part. That’s why we call it RLHF. The following diagram shows how the reward function is built for ChatGPT.

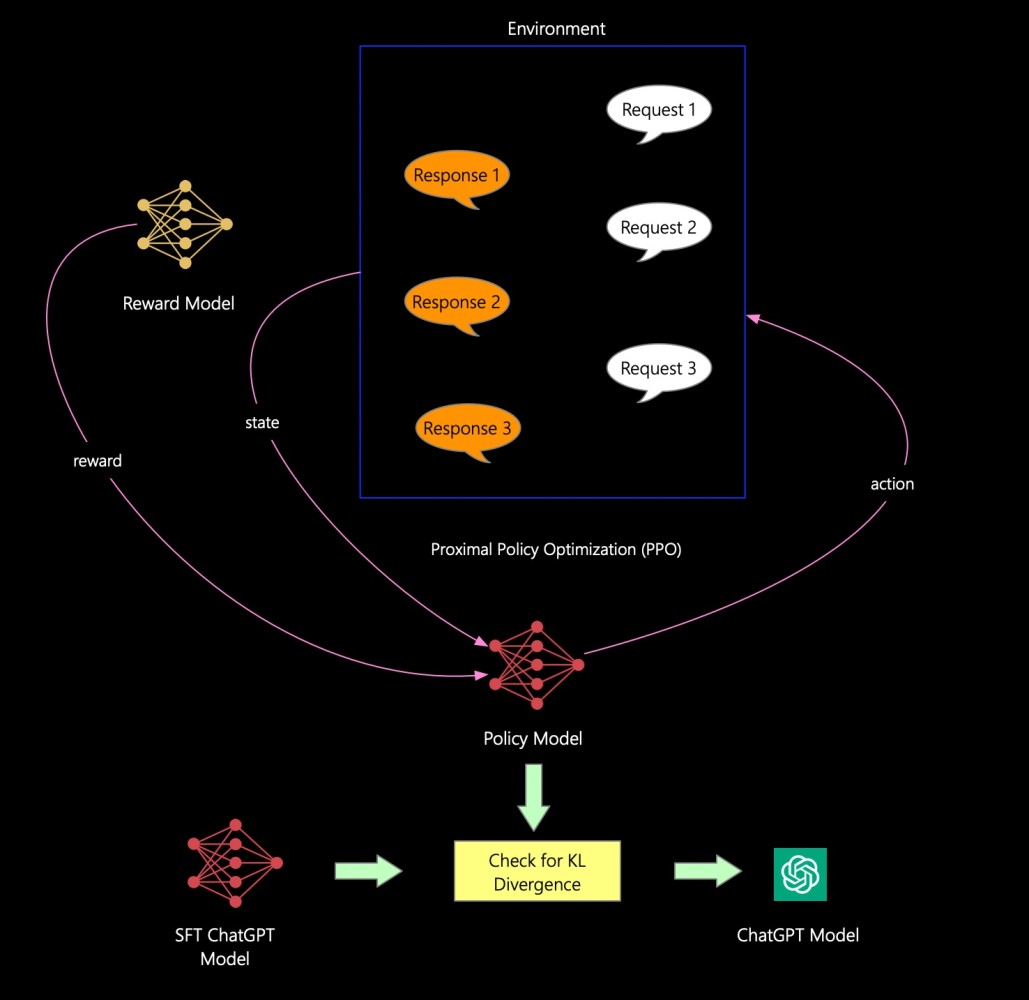


The reward function is established using the following steps:

1. A real person chats with ChatGPT that was trained with some serious smarts at the SFT stage. Then, we mix it up by trying out different alternate responses.
2. Next up, another real person decides which responses they like best by ranking them from top to bottom.
3. In the context of a conversation, training pairs consist of a request and a response. These pairs help the reward model learn what responses are better than others. The reward model gives a high score to ChatGPT when its response is really good compared to the other responses. The reward model is initialized with the same weights as the SFT model.

The reward model spits out scores for each response. The bigger the score, the more likely the model thinks that response is preferred. The reward model is like a binary classifier that uses standard cross-entropy as its loss function. Cross-entropy is just a way to measure the difference between two probability distributions and is pretty common in classification tasks where you’re trying to predict the class of something based on some features. Basically, the cross-entropy loss function will punish the model more when it makes predictions that are way off from what they should be. The whole point of this model is to make the cross-entropy loss as low as possible during training so that it’ll be better at predicting stuff it hasn’t seen before.

Okay, we’ve only trained the reward model for now. Then we use that to do some reinforcement learning. Check out the diagram below to see how we keep using the reward model and policy model to fine-tune the ChatGPT model even more.



So, here’s the deal: the reward model is in charge of giving reward scores to ChatGPT’s answers, while the policy model is all about ChatGPT’s own model. The training process is done interactively and uses reinforcement learning. Basically, given a certain situation (like the history of the conversation), each action (like, what ChatGPT says next) is evaluated through a reward model that uses Proximal Policy Optimization (PPO), which is a fancy algorithm that helps decide what’s a good response and what’s not. PPO works by updating the policy function in small steps so that it gets better and better at choosing the best response. To do this, the algorithm uses something called the “advantage function,” which basically measures how much better one response is compared to all the other possible responses. By updating the policy function in small, “proximal” steps, PPO makes sure that ChatGPT doesn’t make any huge mistakes and stays on track to give great answers.

The RLHF stage isn’t quite there yet. The model trained with PPO is just a guess at what we want. Right now, the main problem is overthinking a.k.a over-optimizing, which is when the reward gives better scores even though the model is doing things we don’t want. Basically, the model is taking advantage of the reward model not being perfect.

A slight deviation: This phenomenon is where people start messing with the measure used to evaluate their progress. Yeah, that’s called Goodhart’s Law. It is a principle that states:

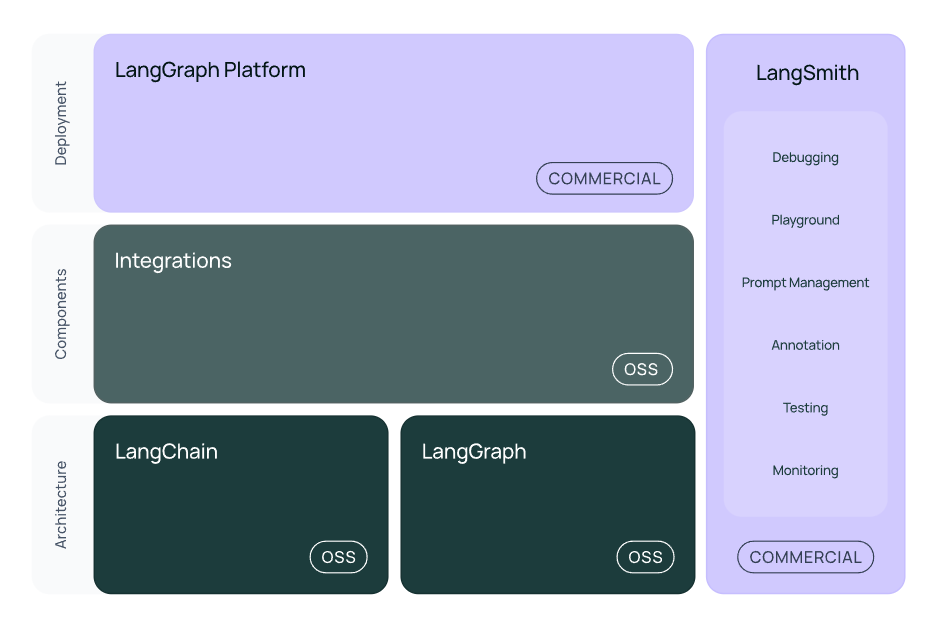
“when a measure becomes a target, it ceases to be a good measure.”

If someone’s incentivized to achieve a certain goal, they might end up distorting the measure and causing unintended consequences. The dude who came up with this is Charles Goodhart, an economist who was talking about monetary policy.

ChatGPT had to deal with this issue, and they totally fixed it. It was kind of a big deal. They added some extra thing to the PPO model, called the KL divergence or Kullback-Leibler divergence, which is like this measure of difference between two probability distributions. Basically, it tells you how much info gets lost when one thing is used to guess the other. Which is pretty useful in machine learning, actually. In machine learning, it is commonly used for tasks such as clustering, anomaly detection, and generative modeling. To keep the PPO model working great, they make it get in trouble if the KL divergence is too high between the RL policy and the SFT’s fine-tuning. The SFT is what the model looks like after Stage 2 of training, just so you know.

Once ChatGPT’s done with this final piece, its model is all set to go and it’s gonna be mind-blowing!

**Langchain Ecosystem:**

****

* Langsmith – LLMOps
* LangGaph – Agentic
* Langchain – Framework for building AI App

