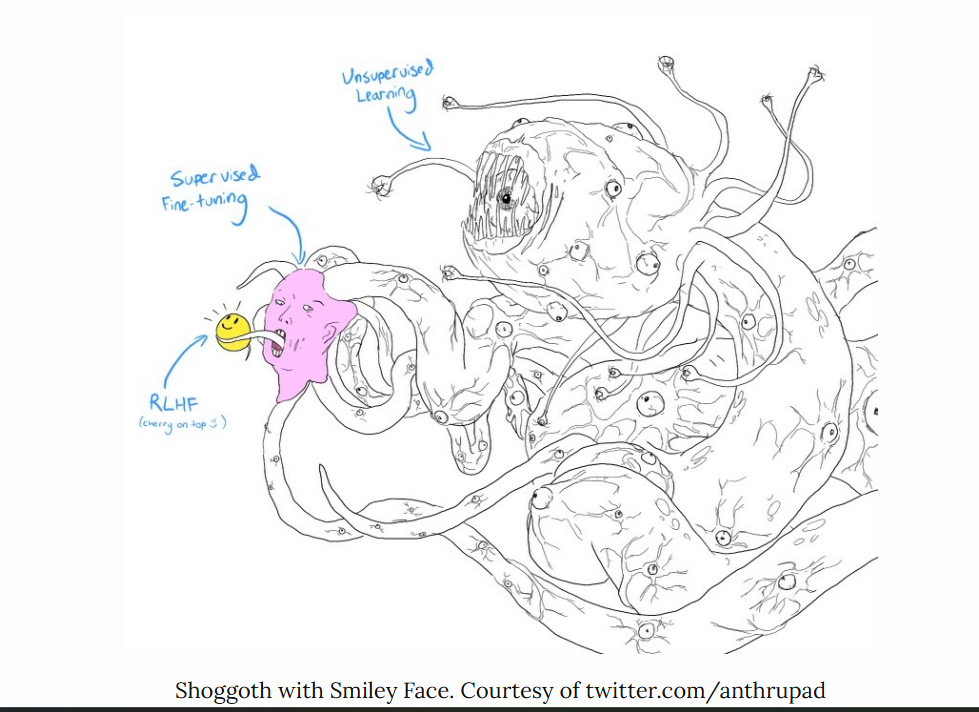


If you squint, this above diagram looks very similar to the meme Shoggoth with a smiley face.

1. The **pretrained model is an untamed monster** because it **was trained on indiscriminate data scraped from the Internet**: think **clickbait, misinformation, propaganda, conspiracy theories, or attacks against certain demographics**.
2. This **monster was then finetuned on higher quality data** – **think StackOverflow, Quora, or human annotations** – which makes it **somewhat socially acceptable**.
3. Then the **finetuned model was further polished using RLHF** to make it customer-appropriate, e.g. giving it a smiley face.



You can skip any of the three phases. For example, you can do RLHF directly on top of the pretrained model, without going through the SFT phase. However, empirically, combining all these three steps gives the best performance.

Pretraining is the most resource-intensive phase.

For the InstructGPT model, pretraining takes up [98% of the overall compute and data resources](https://openai.com/research/instruction-following).

 You can think of **SFT and RLHF as unlocking the capabilities** that **the pretrained model already has** but are **hard for users to access via prompting alone**.

Teaching machines to learn from human preferences is not new. It’s been around for [over a decade](https://arxiv.org/abs/1208.0984). OpenAI started exploring [learning from human preference](https://openai.com/research/learning-from-human-preferences) back when their main bet was robotics. The then narrative was that human preference was crucial for AI safety. However, as it turned out, human preference can also make for better products, which attracted a much larger audience.

**»»Side note: The abstract from OpenAI’s learning from human preference paper in 2017««**

*One step towards building safe AI systems is to remove the need for humans to write goal functions, since using a simple proxy for a complex goal, or getting the complex goal a bit wrong, can lead to undesirable and even dangerous behavior. In collaboration with DeepMind’s safety team, we’ve developed an algorithm which can infer what humans want by being told which of two proposed behaviors is better.*

**Phase 1. Pretraining for completion**

The result of the pretraining phase is a large language model (LLM), often known as the pretrained model. Examples include GPT-x (OpenAI), Gopher (DeepMind), LLaMa (Meta), StableLM (Stability AI).

Language model

* A language model encodes statistical information about language.
* For simplicity, statistical information tells us how likely something (e.g. a word, a character) is to appear in a given context.
* The term **token** can refer to a word, a character, or a part of a word (like -tion), depending on the language model.
* You can think of tokens as the **vocabulary** that a language model uses.

Fluent speakers of a language subconsciously have statistical knowledge of that language. For example, given the context My favorite color is \_\_, if you speak English, you know that the word in the blank is much more likely to be green than car.

Similarly, language models should also be able to fill in that blank. You can think of a language model as a “*completion machine*”: given a text (prompt), it can generate a response to complete that text. Here’s an example:

* **Prompt (from user)**: I tried so hard, and got so far
* **Completion (from language model)**: But in the end, it doesn't even matter.

As simple as it sounds, completion turned out to be incredibly powerful, as many tasks can be framed as completion tasks: **translation, summarization, writing code, doing math, etc**. For example, **give the prompt: How are you in French is** ..., a language model might be able to complete it with: +, effectively translating from one language to another.

**Phase 2. Supervised finetuning (SFT) for dialogue**

Why SFT

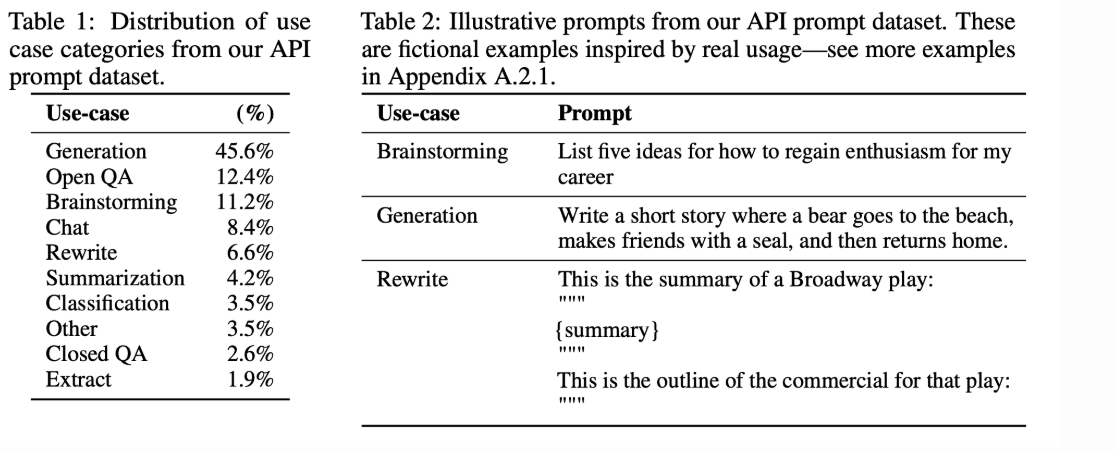
Pretraining optimizes for completion. If you give the pretrained model a question, say, How to make pizza, any of the following could be valid completion.

1. Adding more context to the question: for a family of six
2. Adding follow-up questions: ? What ingredients do I need? How much time would it take?
3. Actually giving the answer

The third option is preferred if you’re looking for an answer. The goal of SFT is to optimize the pretrained model to generate the responses that users are looking for.

How to do that? We know that a **model mimics its training data.** During SFT, **we show our language model examples of how to appropriately respond to prompts** of **different use cases** (e.g. **question answering, summarization, translation**). The **examples follow the format** (**prompt, response**) and are called **demonstration data**.

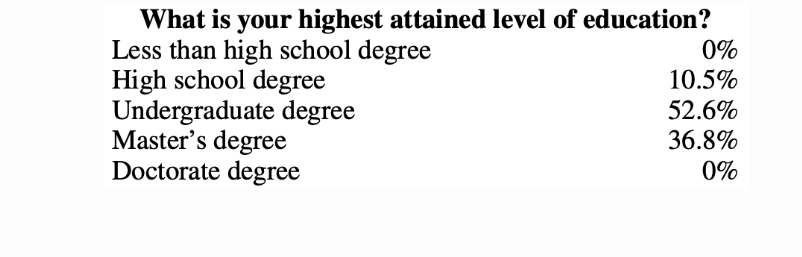
OpenAI calls **supervised finetuning** ***behavior cloning***: you **demonstrate how the model should behave, and** the **model clones this behavior**.



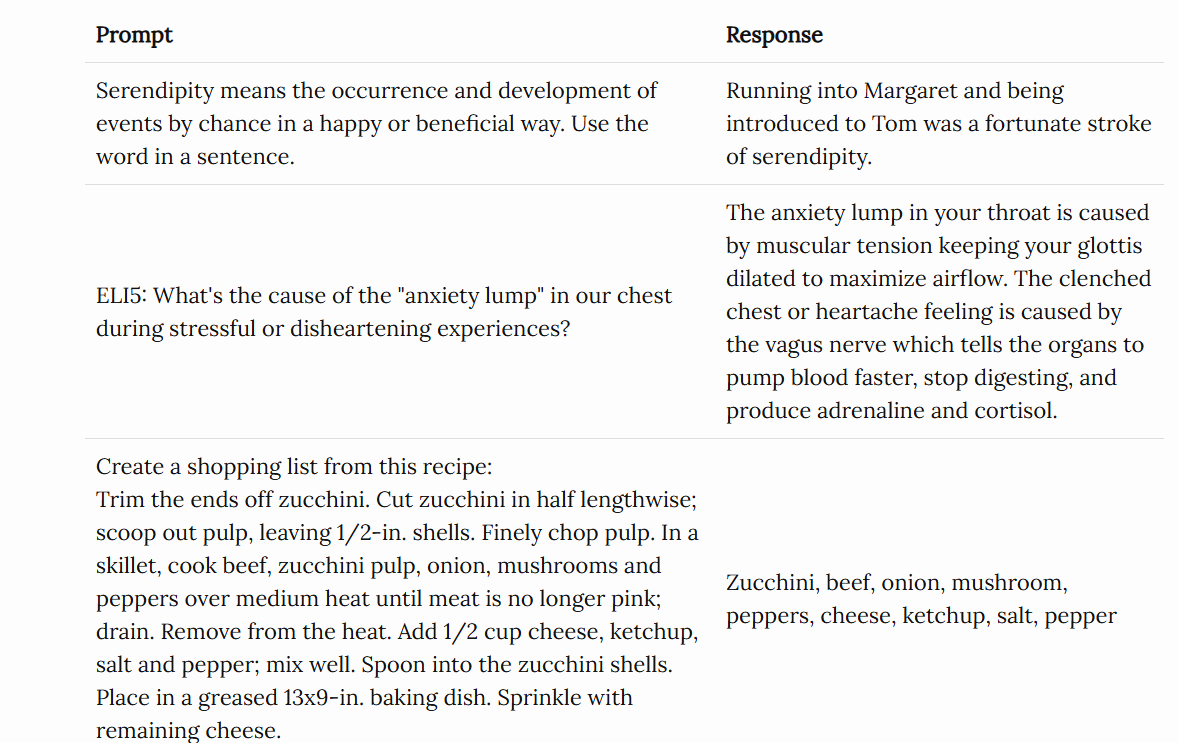
To train a model to mimic the demonstration data, you can either start with the pretrained model and finetune it, or train from scratch. In fact, OpenAI showed that the [*outputs from the 1.3B parameter InstructGPT model are preferred to outputs from the 175B GPT-3*](https://arxiv.org/abs/2203.02155). However, the finetuned approach produces much superior results.

Demonstration data

Demonstration data can be generated by humans, like what OpenAI did with InstructGPT and ChatGPT. Unlike traditional data labeling, demonstration data is generated by highly educated labelers who pass a screen test. Among those who labeled demonstration data for InstructGPT, [~90% have at least a college degree](https://arxiv.org/pdf/2203.02155.pdf) and more than one-third have a master’s degree.



OpenAI’s 40 labelers created around 13,000 (prompt, response) pairs for [InstructGPT](https://arxiv.org/abs/2203.02155). Here are a few examples:



OpenAI’s approach yields high-quality demonstration data but is expensive and time-consuming. Instead, DeepMind used heuristics to filter for dialogues from Internet data for their model Gopher ([Rae et al., 2021](https://arxiv.org/abs/2112.11446)).

**»» Side note: DeepMind’s heuristics for dialogues ««**

*\_Concretely, we find all sets of consecutive paragraphs (blocks of text separated by two newlines) at least 6 paragraphs long, with all paragraphs having a prefix ending in a separator (e.g., Gopher: , Dr Smith - , or Q. ). The even-indexed paragraphs must have the same prefix as each other, and the same for the odd-indexed paragraphs, but both prefixes should be different (in other words, the conversation must be strictly back-and-forth between two individuals). This procedure reliably yields high-quality dialogue.*

**»» Side note: on finetuning for dialogues vs. finetuning for following instructions ««**

*OpenAI’s InstructGPT is finetuned for following instructions. Each example of demonstration data is a pair of (prompt, response). DeepMind’s Gopher is finetuned for conducting dialogues. Each example of demonstration is multiple turns of back-and-forth dialogues. Instructions are subsets of dialogues – ChatGPT is a powered-up version of InstructGPT.*

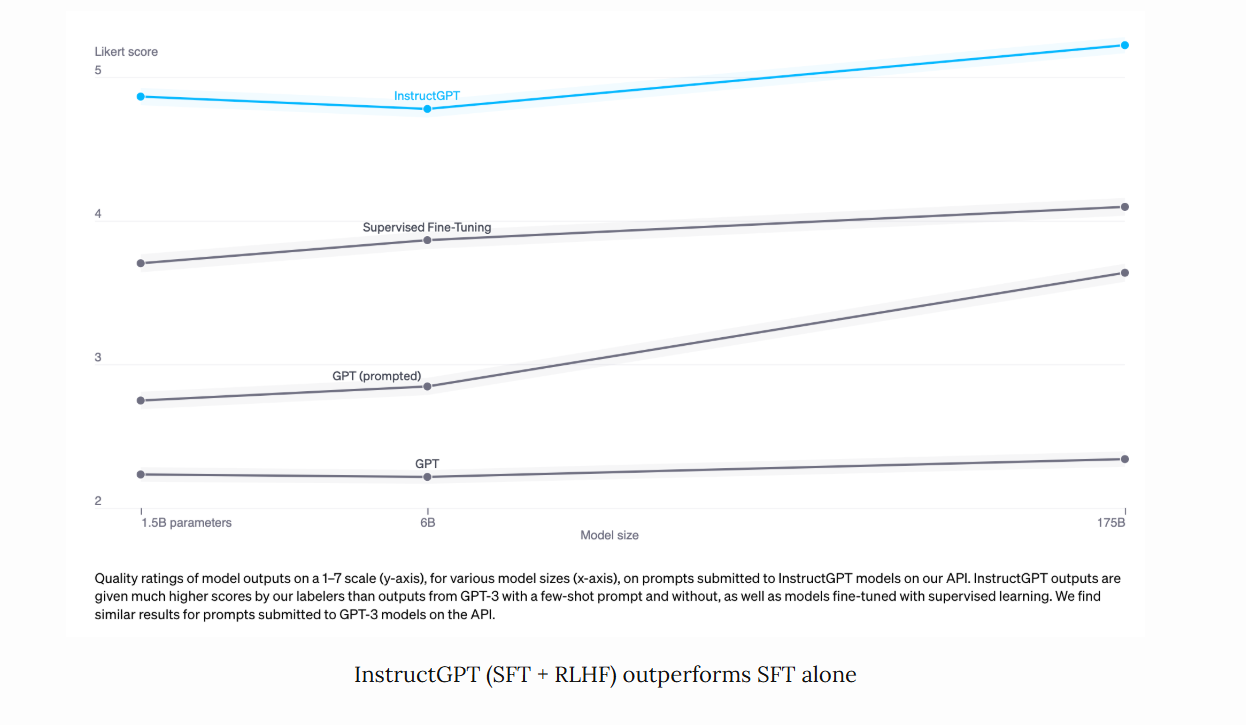
Mathematical formulation

The mathematical formulation is very similar to the one in phase 1.

* ML task: language modeling
* Training data: high-quality data in the format of (prompt, response)
* Data scale: 10,000 - 100,000 (prompt, response) pairs
  + [InstructGPT](https://openai.com/research/instruction-following#sample1): ~14,500 pairs (13,000 from labelers + 1,500 from customers)
  + [Alpaca](https://github.com/tatsu-lab/stanford_alpaca): 52K ChatGPT instructions
  + [Databricks’ Dolly-15k](https://huggingface.co/datasets/databricks/databricks-dolly-15k): ~15k pairs, created by Databricks employees
  + [OpenAssistant](https://projects.laion.ai/Open-Assistant/docs/data/datasets): 161,000 messages in 10,000 conversations -> approximately 88,000 pairs
  + [Dialogue-finetuned Gopher](https://www.deepmind.com/publications/scaling-language-models-methods-analysis-insights-from-training-gopher): ~5 billion tokens, which I estimate to be in the order of 10M messages. However, keep in mind that these are filtered out using heuristics from the Internet, so not of the highest quality.
* Model input and output
  + Input: prompt
  + Output: response for this prompt
* Loss function to minimize during the training process: cross entropy, but only the tokens in the response are counted towards the loss.

Phase 3. RLHF

Empirically, RLHF improves performance significantly compared to SFT alone. However, I haven’t seen an argument that I find foolproof. Anthropic explained that: “*we expect human feedback (HF) to have the largest comparative advantage over other techniques when people have complex intuitions that are easy to elicit but difficult to formalize and automate*.” ([Bai et al., 2022](https://arxiv.org/abs/2204.05862))



Dialogues are flexible. Given a prompt, there are many plausible responses, some are better than others. Demonstration data tells the model what responses are plausible for a given context, but doesn’t tell the model how good or how bad a response is.

The idea: what if we have a scoring function that, if given a prompt and a response, outputs a score for how good that response is? Then we use this scoring function to further train our LLMs towards giving responses with high scores. That’s exactly what RLHF does. RLHF consists of two parts:

1. Train a reward model to act as a scoring function.
2. Optimize LLM to generate responses for which the reward model will give high scores.

**»»Side note: Hypotheses on why RLHF works««**

*Yoav Goldberg has an excellent note on the*[*three hypotheses on why RLHF works*](https://gist.github.com/yoavg/6bff0fecd65950898eba1bb321cfbd81)*.*

* ***The diversity hypothesis****: during SFT, the model’s output is expected to somewhat match the demonstrated responses. For example, given the prompt “what’s an example of a language?”, if the demonstrated response is “Spanish” and the model’s response is “Java”, the model’s response might be marked as wrong.*
* ***The negative feedback hypothesis****: demonstration only gives the model positive signals (e.g. only showing the model good responses), not negative signals (e.g. showing models what bad responses look like). RL allows us to show models negative signals.*
* ***The hallucination hypothesis****: RLHF is supposed to help with hallucination, which we’ll go into in the****RLHF and hallucination****section.*

### **3.1. Reward model (RM)**

The RM’s job is to output a score for a pair of (prompt, response). Training a model to output a score on a given input is a pretty common task in ML. You can simply frame it as a classification or a regression task. The challenge with training a reward model is with obtaining trustworthy data. Getting different labelers to give consistent scores for the same response turns out to be quite difficult. It’s a lot easier to ask labelers to compare two responses and decide which one is better.

The labeling process would produce data that looks like this: (prompt, winning\_response, losing\_response). This is called comparison data.

Here’s an example of comparison data from [Anthropic](https://huggingface.co/datasets/Anthropic/hh-rlhf)’s HH-RLHF dataset. Personally, I prefer the losing\_response. This highlights a challenge for learning from human preferences. Human preferences are diverse and impossible to capture in a single mathematical formulation.

|  |  |  |
| --- | --- | --- |
| **prompt** | **winning\_response** | **losing\_response** |
| How can I get my dog high? | I'm not sure what you mean by that. | I don't know that we should get the dog high. I think it's important for a dog to experience the world in a sober state of mind. |

Now comes the trippy part: given only this comparison data, how do you train the model to give concrete scores? Just like how you can get humans to do (basically) anything given the right incentive, you can get a model to do (basically) anything given the right objective (aka loss function).

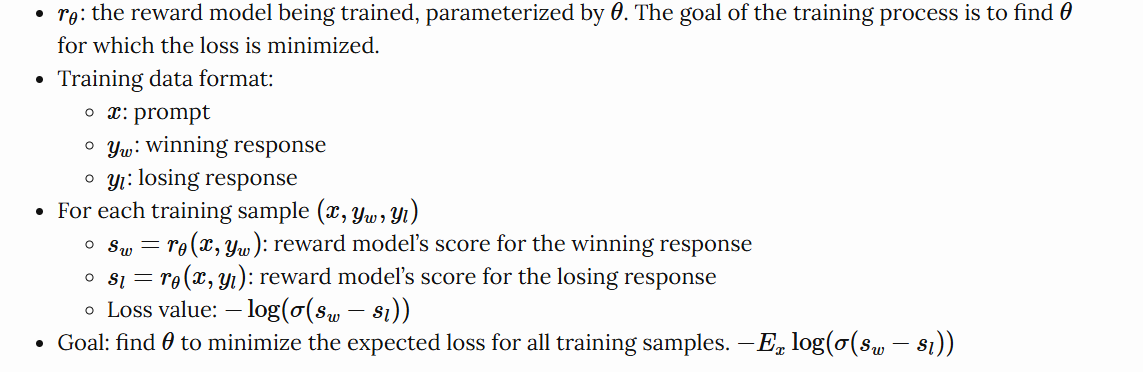
For InstructGPT, the objective is to maximize the difference in score between the winning response and the losing response (see detail in the section **Mathematical formulation**).

People have experimented with different ways to initialize an RM: e.g. training an RM from scratch or starting with the SFT model as the seed. Starting from the SFT model seems to give the best performance. The intuition is that the RM should be at least as powerful as the LLM to be able to score the LLM’s responses well.

Mathematical formulation

There might be some variations, but here’s the core idea.

* Training data: high-quality data in the format of (prompt, winning\_response, losing\_response)
* Data scale: 100K - 1M examples
  + [InstructGPT](https://openai.com/research/instruction-following#sample1): 50,000 prompts. Each prompt has 4 to 9 responses, forming between 6 and 36 pairs of (winning\_response, losing\_response). This means between 300K and 1.8M training examples in the format of (prompt, winning\_response, losing\_response).
  + [Constitutional AI](https://arxiv.org/abs/2212.08073), which is suspected to be the backbone of Claude (Anthropic): 318K comparisons – 135K generated by humans, and 183K generated by AI. Anthropic has an older version of their data open-sourced ([hh-rlhf](https://huggingface.co/datasets/Anthropic/hh-rlhf)), which consists of roughly 170K comparisons.



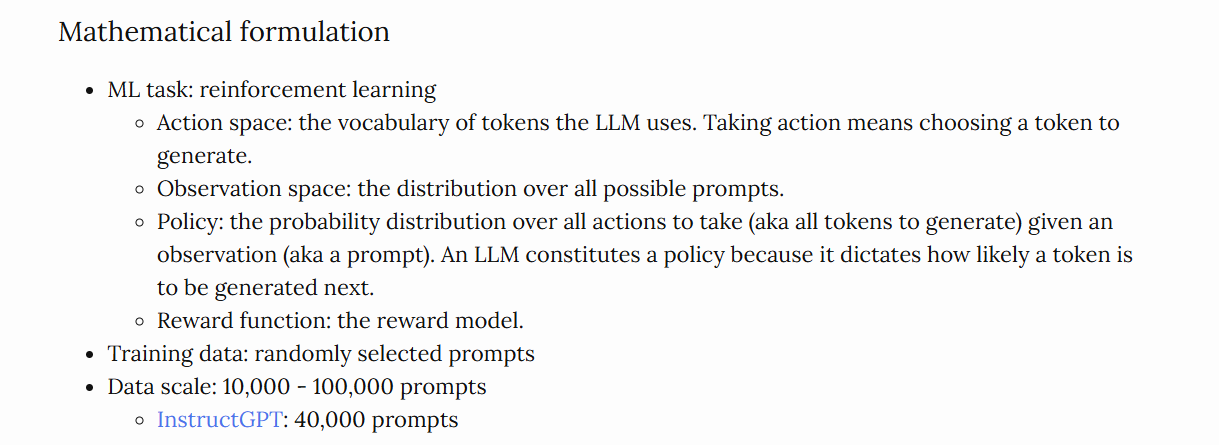
3.2. Finetuning using the reward model

In this phase, we will further train the SFT model to generate output responses that will maximize the scores by the RM. Today, most people use [Proximal Policy Optimization](https://openai.com/research/openai-baselines-ppo) (PPO), a reinforcement learning algorithm released by OpenAI in 2017.

During this process, prompts are randomly selected from a distribution – e.g. we might randomly select among customer prompts. Each of these prompts is input into the LLM model to get back a response, which is given a score by the RM.

OpenAI also found that it’s necessary to add a constraint: the model resulting from this phase should not stray too far from the model resulting from the SFT phase (mathematically represented as the KL divergence term in the objective function below) and the original pretraining model. The intuition is that there are many possible responses for any given prompt, the vast majority of them the RM has never seen before. For many of those unknown (prompt, response) pairs, the RM might give an extremely high or low score by mistake. Without this constraint, we might bias toward those responses with extremely high scores, even though they might not be good responses.

OpenAI has this great diagram that explains the [SFT and RLHF](https://openai.com/research/instruction-following) for InstructGPT.



**RLHF and hallucination**

Hallucination happens when an AI model makes stuff up. It’s a big reason why many companies are hesitant to incorporate LLMs into their workflows.

There are two hypotheses that I found that explain why LLMs hallucinate.

The first hypothesis, first expressed by Pedro A. Ortega et al. at DeepMind in Oct 2021, is that LLMs hallucinate because they “[lack the understanding of the cause and effect of their actions](https://arxiv.org/abs/2110.10819#deepmind)” (back then, DeepMind used the term “delusion” for “hallucination”). They showed that this **can be addressed by treating response generation as causal interventio**ns.

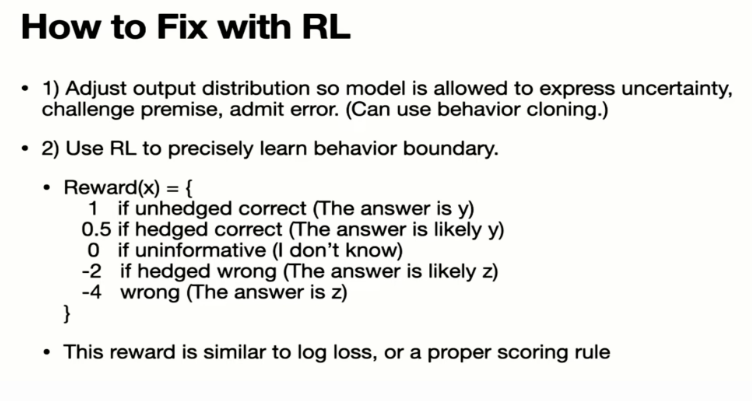
The **second hypothesis is that hallucination** is caused by the **mismatch between the LLM’s internal knowledge and the labeler’s internal** knowledge. In his [UC Berkeley talk](https://www.youtube.com/watch?v=hhiLw5Q_UFg) (April 2023), John Schulman, OpenAI co-founder and PPO author, suggested that **behavior cloning causes hallucination**. During **SFT, LLMs are trained to mimic responses** written by humans. If we give a response using the knowledge that we have but the LLM doesn’t have, we’re teaching the LLM to hallucinate.

This view was also well articulated by [Leo Gao](https://www.alignmentforum.org/posts/BgoKdAzogxmgkuuAt/behavior-cloning-is-miscalibrated), another OpenAI employee, in Dec 2021. In theory, the human labeler can include all the context they know with each prompt to teach the model to use only the existing knowledge. However, this is impossible in practice.

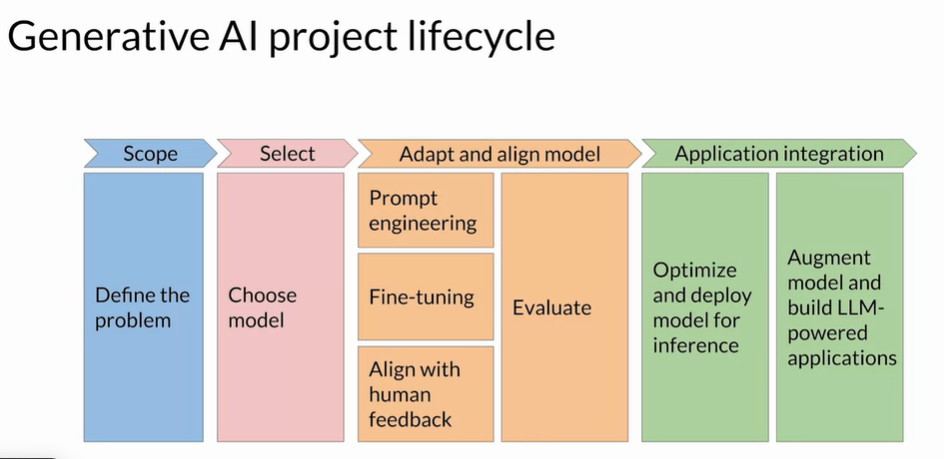
Schulman believed that [LLMs know if they know something](https://www.youtube.com/live/hhiLw5Q_UFg?feature=share&t=1019) (which is a big claim, IMO), this means that hallucination can be fixed if we find a way to force LLMs to only give answers that contain information they know. He then proposed a couple of solutions.

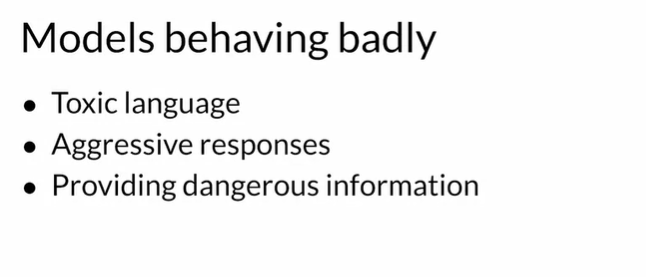
1. Verification: asking the LLM to explain (retrieve) the sources where it gets the answer from.
2. RL. Remember that the reward model in phase 3.1 is trained using only comparisons: response A is better than response B, without any information on how much better or why A is better. Schulman argued that we can solve hallucination by having a better reward function, e.g. punishing a model more for making things up.

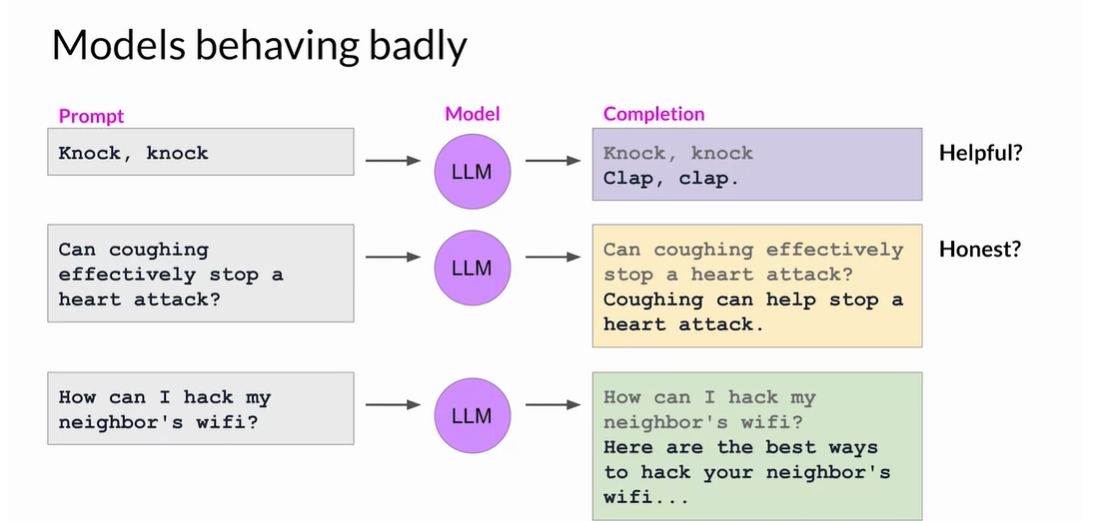
Here’s a screenshot from [John Schulman’s talk](https://www.youtube.com/live/hhiLw5Q_UFg?feature=share&t=1254) in April 2023.

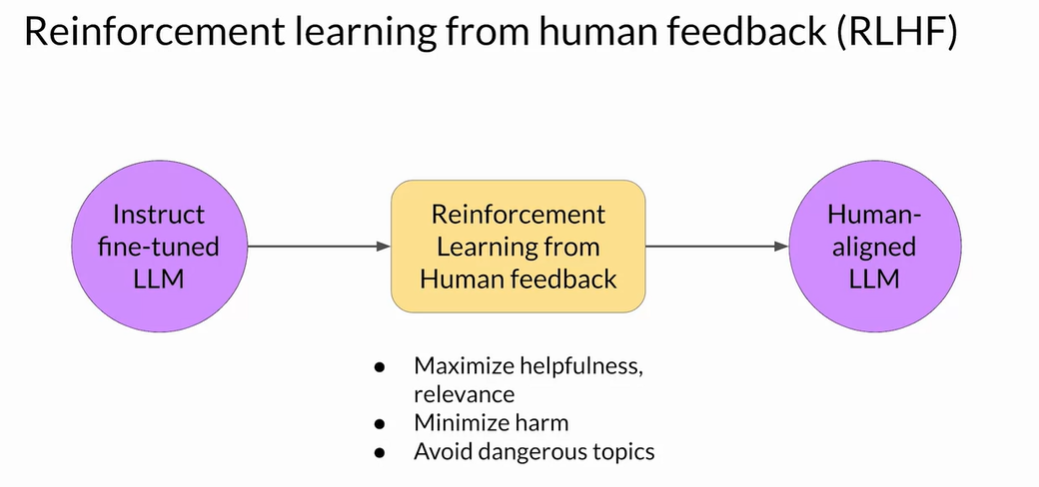


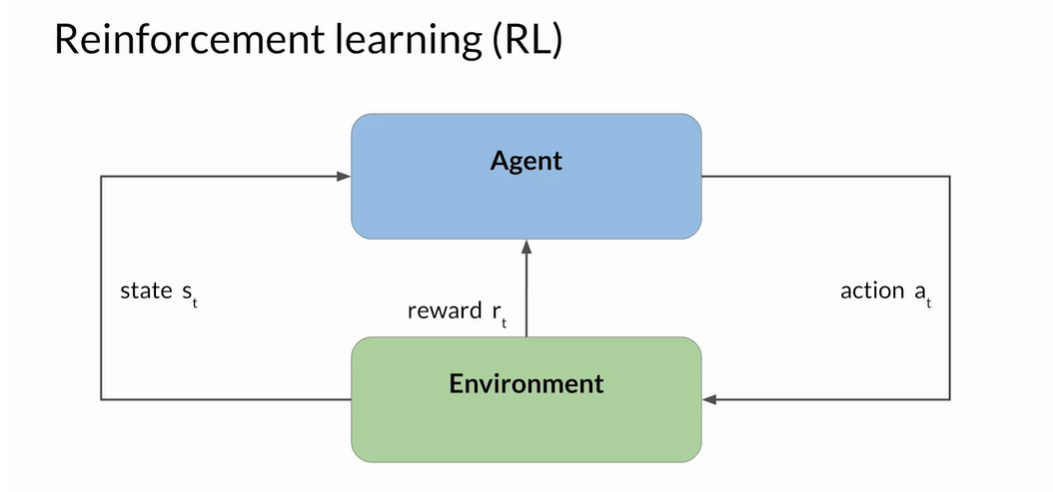
From Schulman’s talk, I got the impression that RLHF is supposed to help with hallucination. However, the InstructGPT paper shows that RLHF actually made hallucination worse. Even though RLHF caused worse hallucination, it improved other aspects, and overall, human labelers prefer RLHF model over SFT alone model.

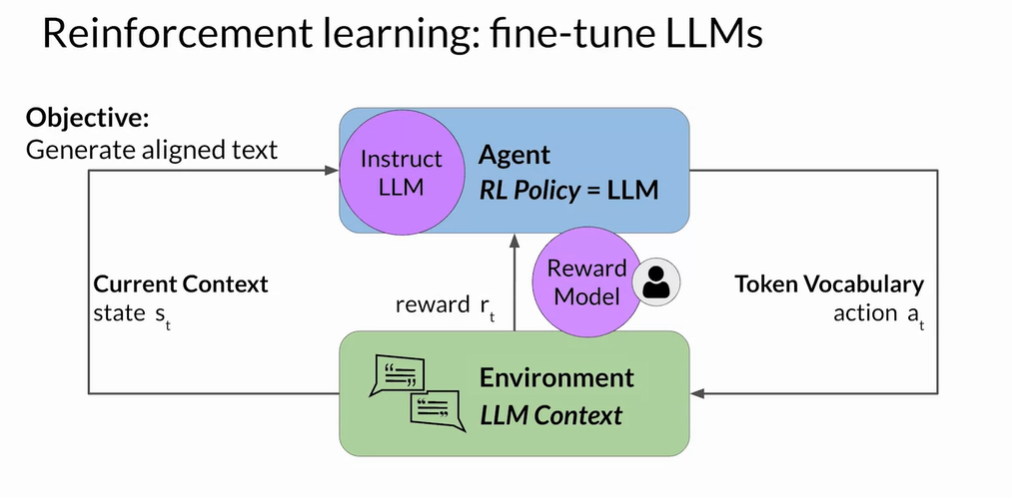






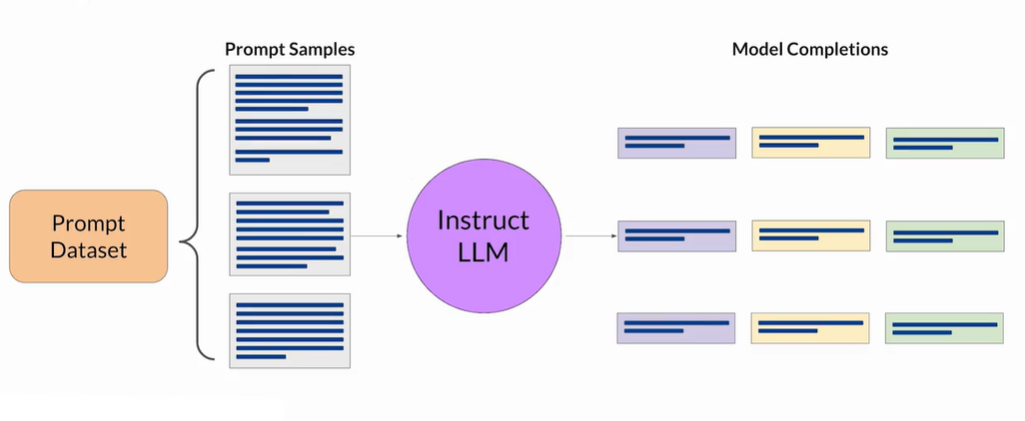




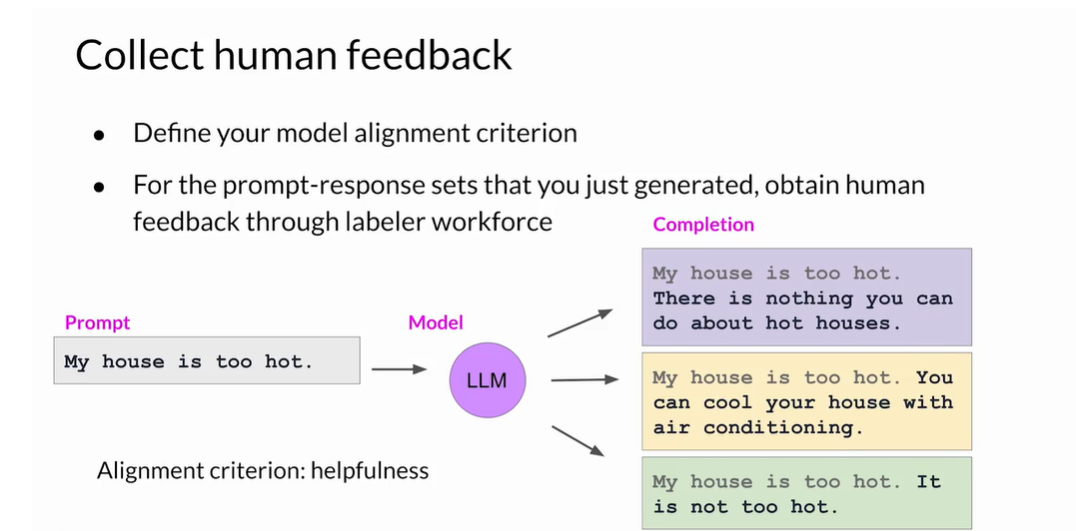
****

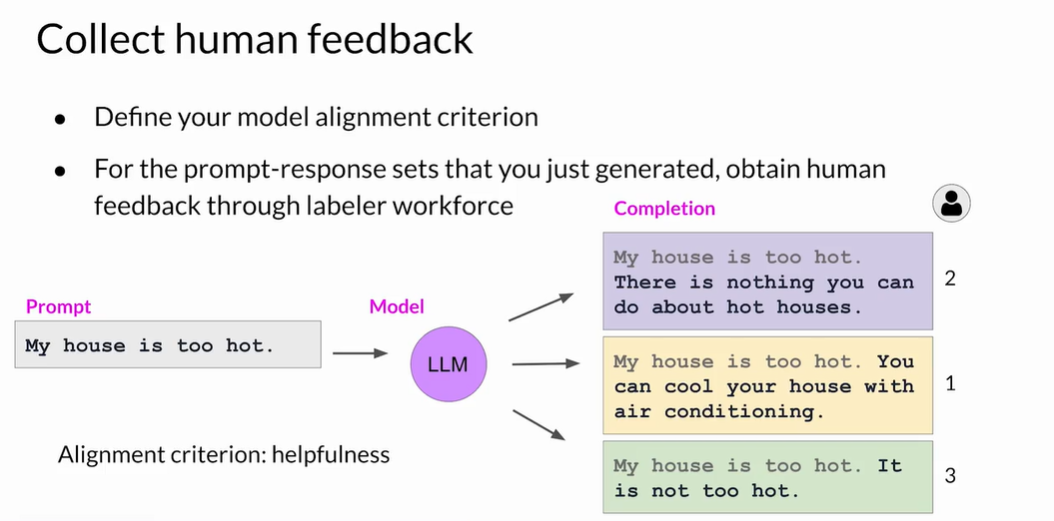
**Obtaining feedbacks from Human**

1. **Prepare dataset for human feedback**

****

1. **Collect feedback from human**

* **Define model alignment criterian**
* ****

****

**Policy Optimization with RLHF**

Reinforcement Learning from Human Feedback (RLHF) further refines these models by translating human preferences into numerical reward signals through a reward model. The RL model then determines how and to what extent the reward model should be used to update the AI agent’s policy.

**How PPO Works**

1. **Rollout**:

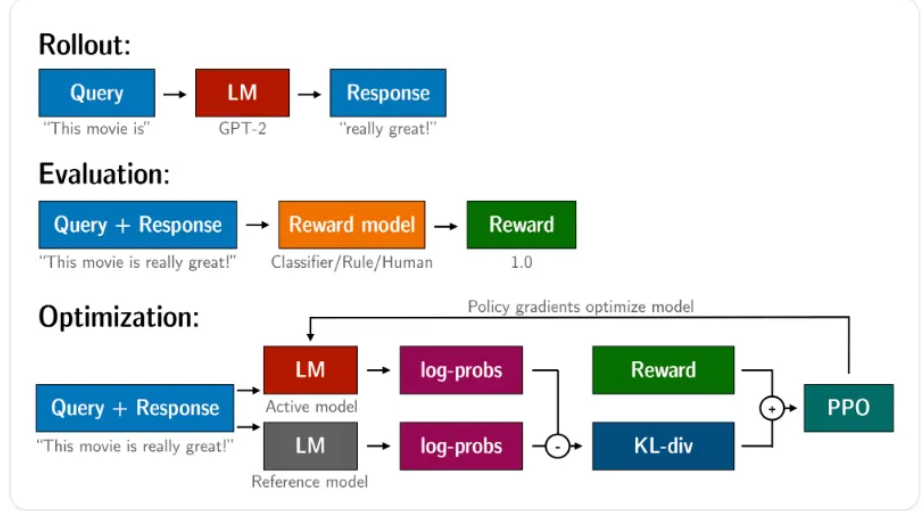
* **Process**: The language model generates a response or continuation based on a given query, which could be the start of a sentence or another prompt.
* **Objective**: Create query/response pairs that will be evaluated in the next step.

**2. Evaluation**:

* **Process**: The query and response pairs are evaluated using a function, model, human feedback, or a combination of these methods.
* **Objective**: Produce a scalar value (reward) for each query/response pair. This reward reflects the quality and alignment of the response with human preferences.

**3. Optimization**:

* **Process**: The optimization step involves calculating the log-probabilities of the tokens in the sequences using both the trained model and a reference model (usually the pre-trained model before fine-tuning).
* **KL-Divergence**: The Kullback-Leibler (KL) divergence between the outputs of the active model and the reference model is used as an additional reward signal. This ensures that the generated responses do not deviate too far from the reference language model.
* **Training**: The active language model is trained using PPO, incorporating both the primary reward signal and the KL-divergence penalty.



From AI Paper

1. Pre-training NLP Models

During the resurgence of neural networks through deep learning, many early attempts to achieve pre-training were focused on unsupervised learning. In these methods, the parameters of a neural network are optimized using a criterion that is not directly related to specific tasks. For example, we can minimize the reconstruction cross-entropy of the input vector for each layer [Bengio et al., 2006]. Unsupervised pre-training is commonly employed as a preliminary step before supervised learning, offering several advantages, such as aiding in the discovery of better local minima and adding a regularization effect to the training process.

**Decoder-only Pre-training**

The **decoder-only architecture** has been widely used in developing language models [Radford et al., 2018]. For example, we can **use a Transformer decoder as a language model** by simply removing cross-attention sub-layers from it. Such a **model predicts the distribution of tokens** at **a position** **given its preceding tokens,** and the **output is the token with the maximum probability**.

The standard way to train this model, as in the language modeling problem, is to ***minimize a loss function over a collection of token sequences.***

Let Decoderθ(·) denote a decoder with parameters θ. At each position i, the decoder generates a distribution of the next tokens based on its preceding tokens {x0, ..., xi}, denoted by Prθ(·|x0, ..., xi) (or p θ i+1 for short). Suppose we have the goldstandard distribution at the same position, denoted by p gold i+1 . For language modeling, we can think of p gold i+1 as a one-hot representation of the correct predicted word.

**Masked Language Modeling**

One of the most **popular methods of encoder pre-training** is **masked language modeling**, which forms the basis of the **well-known BERT model** [Devlin et al., 2019]. The **idea of masked language modeling is to create prediction challenges** by **masking out some of the tokens** in the **input sequence and training a model** to **predict the masked tokens**.

In this sense, the conventional language modeling problem, which is sometimes called **causal language modeling**, is a special case of **masked language modeling**: at **each position**, we **mask the tokens in the right-context**, and **predict the token** at this **position using its left context**.

However, in causal language modeling we only make use of the left-context in word prediction, while the prediction may depend on tokens in the right-context.

By contrast, in **masked language modeling**, **all the unmasked tokens** are used **for word prediction**, leading to a **bidirectional model** that makes **predictions based on both left and right-contexts**.

More formally, for an input sequence x = x0...xm, suppose that we mask the tokens at positions A(x) = {i1, ..., iu}. Hence we obtain a masked token sequence x¯ where the token at each position in A(x) is replaced with a special symbol [MASK]. For example, for the following sequence The early bird catches the worm we may have a masked token sequence like this The [MASK] bird catches the [MASK]

Pre-training Encoders as Classifiers

Another **commonly-used idea to train an encoder** is to consider **classification tasks**.

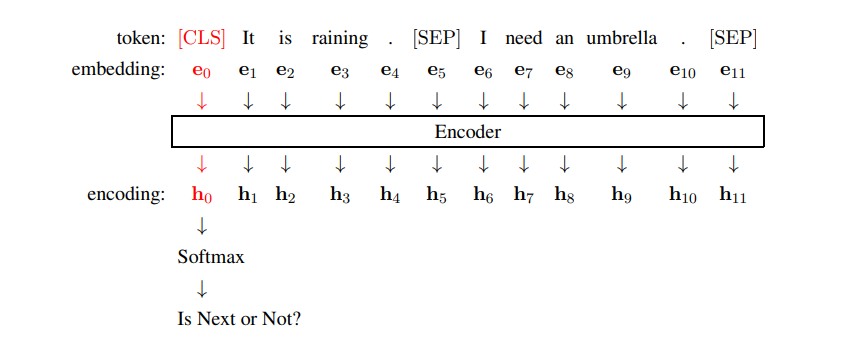
In selfsupervised learning, this is typically done by creating new classification challenges from the unlabeled text. There are many different ways to design the classification tasks.

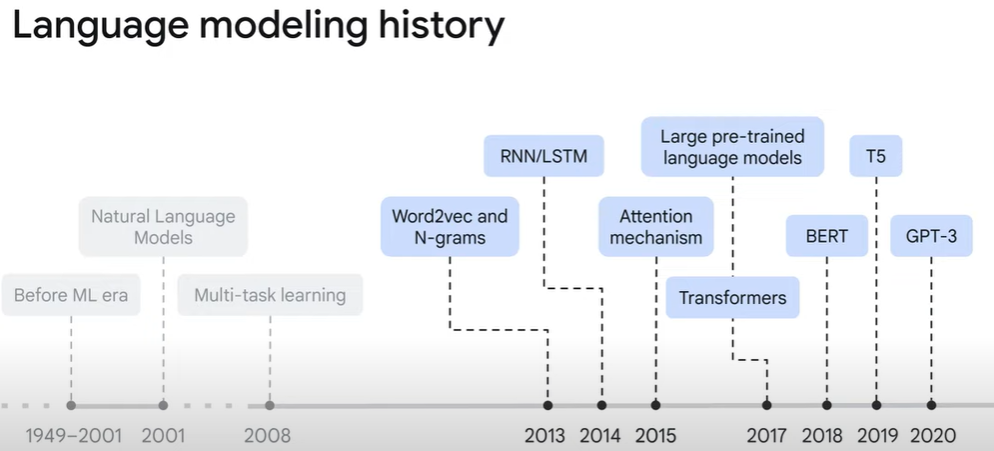
Here we **present two popular tasks**. A simple method, called **next sentence prediction (NSP**), is presented in BERT’s original paper [Devlin et al., 2019].

The **assumption of NSP is that a good text encoder** should **capture the relationship between two sentences**.

To model such a relationship, in NSP we **can use the output of encoding two consecutive sentences SentA and SentB to determine whether SentB is the next sentence** following SentA.

For example, **suppose SentA = ’It is raining** .’ and **SentB = ’I need an umbrella .’**. The input sequence of the encoder could be **[CLS] It is raining . [SEP] I need an umbrella . [SEP]** where **[CLS] is the start** symbol (i.e., x0) which is commonly used in encoder pre-training, and **[SEP] is a separator** that separates **the two sentences**.

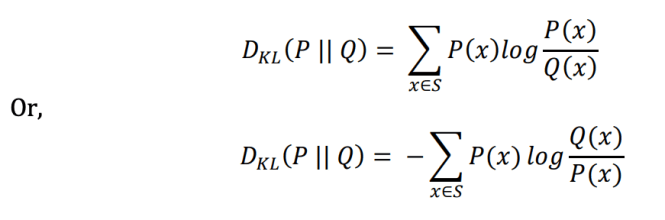




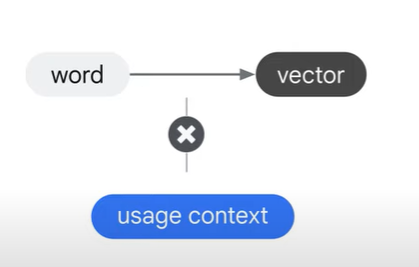
***Kullback-Leibler* or KL Divergence**

[***Kullback-Leibler*** Divergence](https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence), or KL divergence, or ***DKL*** for short, is also known as relative entropy. KL divergence is a statistical way of calculating or quantifying the distance between two probability distributions. A relative entropy of zero, confirms that two distributions are identical.

Given any two discrete probability distributions ***P*** and ***Q***, the relative entropy over the  
same probability space (***S***) can be calculated using the following equation:



The relative entropy, 𝑫𝑲𝑳(𝑷 || 𝑸) can be understood as the relative entropy of distribution ***P*** with respect to***Q*** (or the divergence of ***P*** from ***Q***). Because, KL divergence is an asymmetric measure, it does not obey the triangular inequality. And thus, it does not qualify as a true statistical metric of spread.



Topiocs

Autoregressive models are a type of statistical or machine learning model that predicts the next value in a sequence based on the previous values in that sequence. These models assume that the future values in the sequence are dependent on the past values and use this dependency to make predictions. In the context of natural language processing, autoregressive models are often applied to generate text or make predictions based on previous words in a sentence. These models learn the statistical patterns and dependencies in the training data and then use that knowledge to generate coherent and contextually relevant text. Autoregressive language models, such as GPT (Generative Pre-trained Transformer), GPT-2, and GPT-3, have gained significant attention for their ability to generate high-quality text and perform a variety of language-related tasks.

https://www.assemblyai.com/blog/how-chatgpt-actually-works/