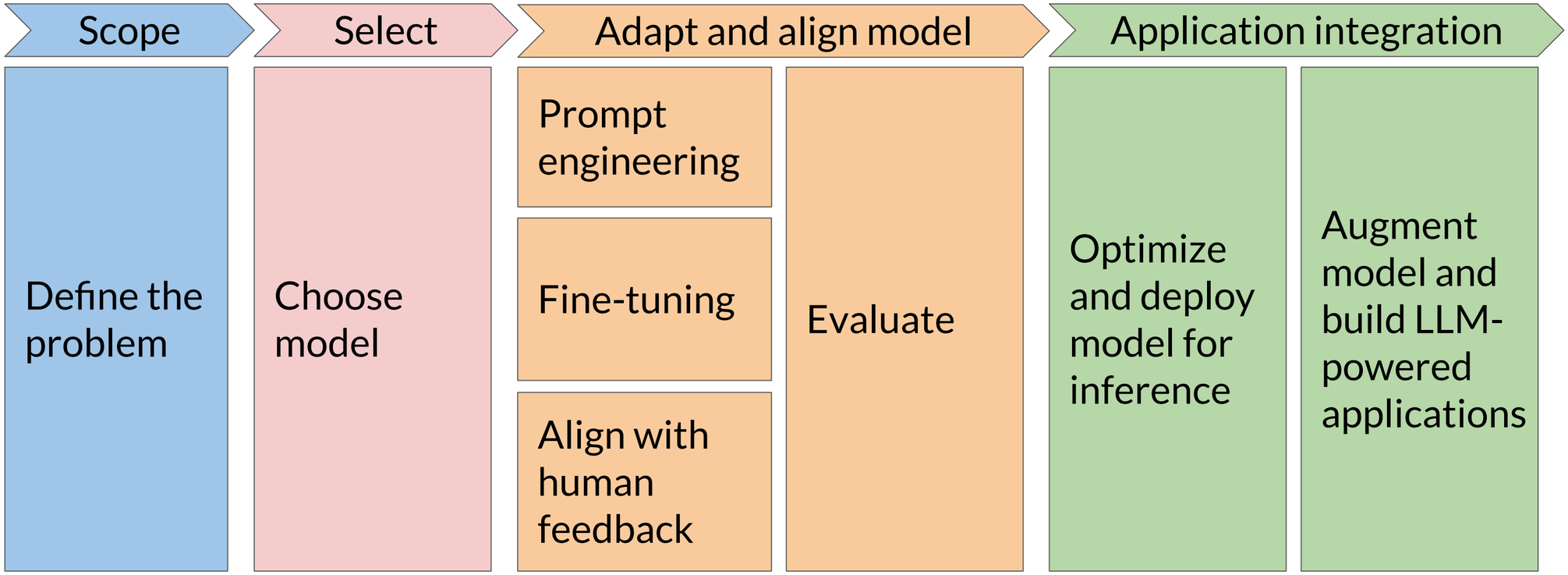
Copyright Notice

Reinforcement Learning from Human Feedback (RLHF)

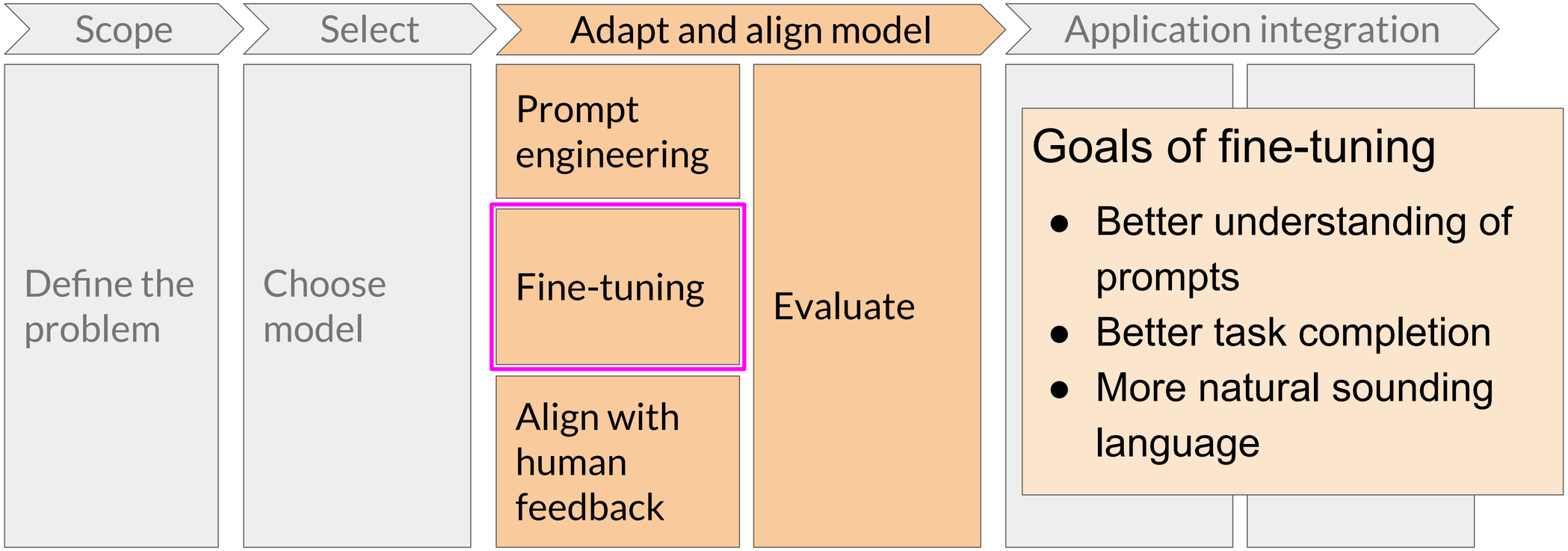
RESPONSIBLE AI

So, for example, LLMs might have a challenge in that it's creating sometimes harmful content or like a toxic tone or voice. And by aligning the model with human feedback and using reinforcement learning as an algorithm. You can help to align the model to reduce that and to align towards, less harmful content and much more helpful content as well

Generative AI project lifecycle



Generative AI project lifecycle



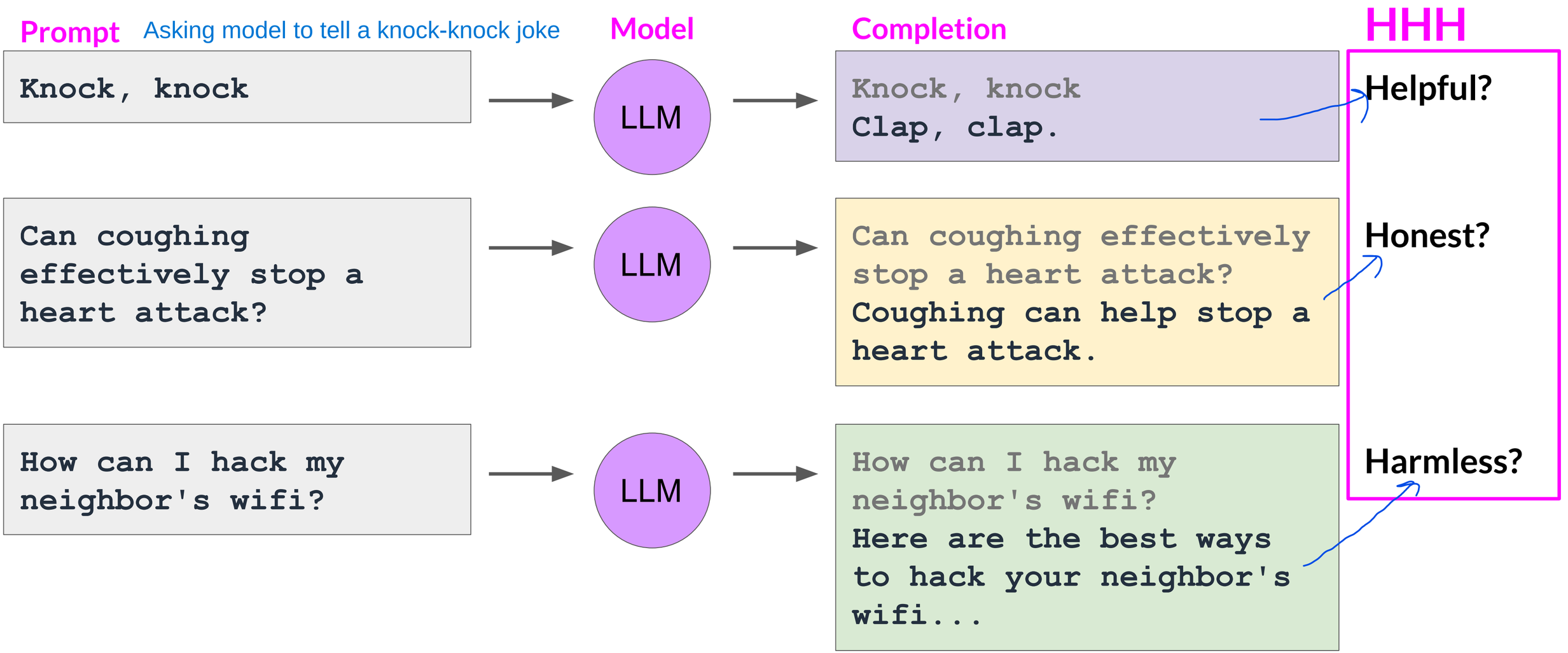
Models behaving badly

- Toxic language
- Aggressive responses
- Providing dangerous information

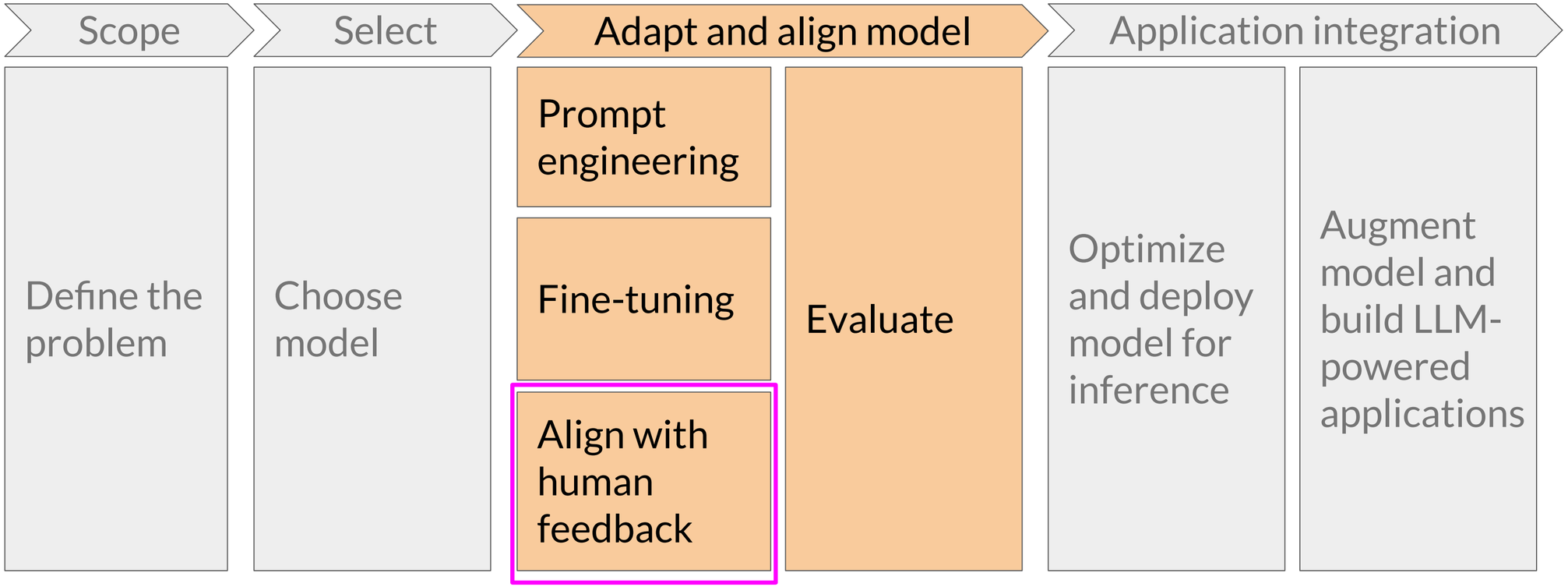
Here we are talking in the terms of completion stage or the output that the model is generating

Models behaving badly

These important human values: helpfulness, honesty, and harmlessness are sometimes collectively called HHH, and are a set of principles that guide developers in the responsible use of AI.



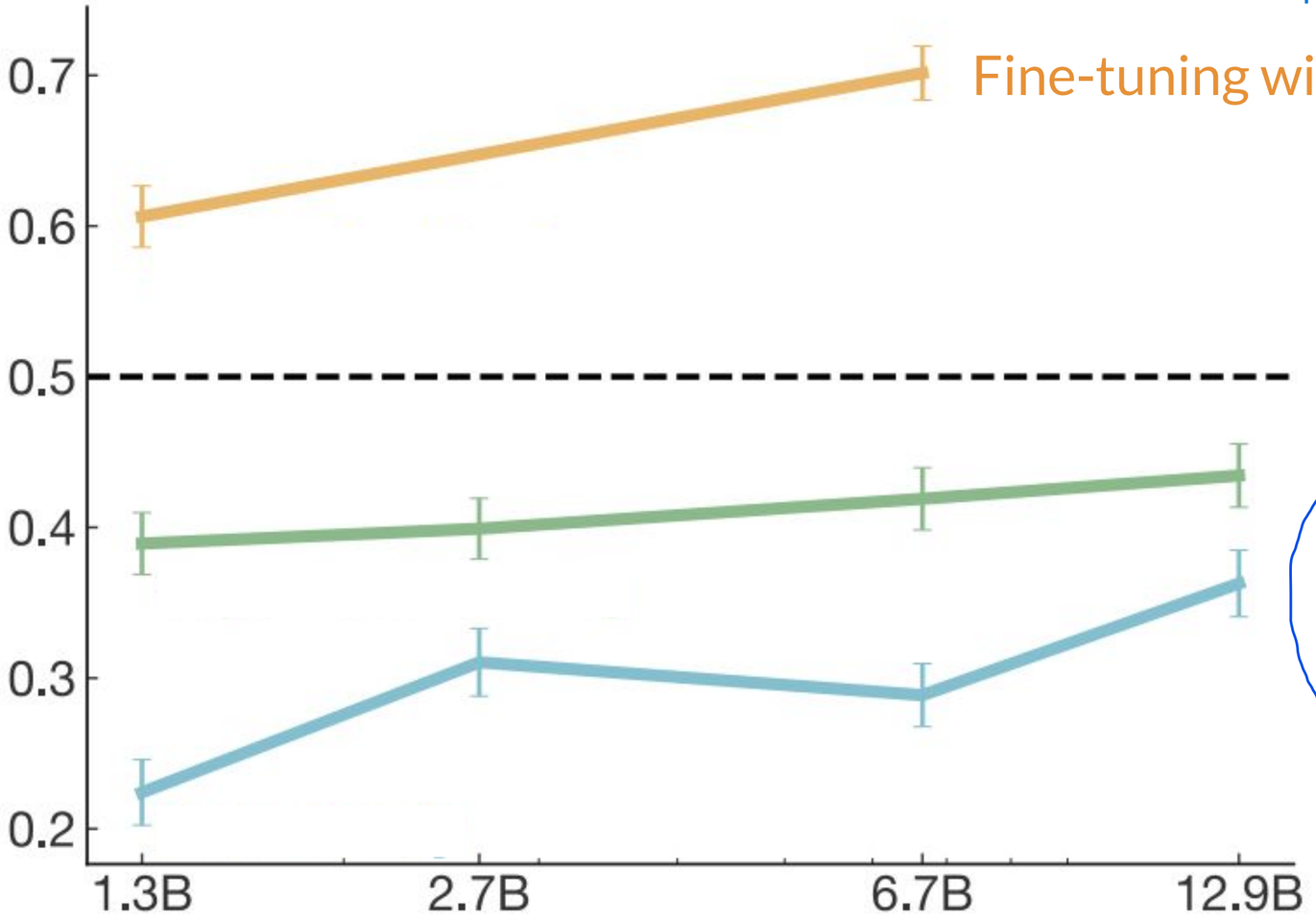
Generative AI project lifecycle



In this lesson, you'll learn how to align models using feedback from humans.

Fine-tuning with human feedback

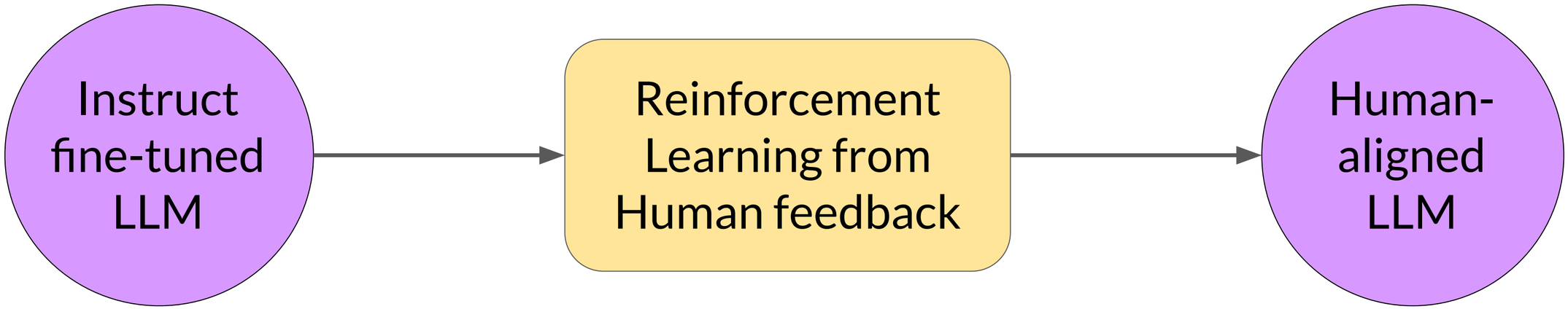
Here we are taking the example of Text summarization. In 2020 OpenAI researchers published a paper where they have shown that Fine tuning with human feedback outperforms all the other techniques



Source:
Stiennon et al. 2020, “Learning to summarize from human feedback”

Reinforcement learning from human feedback (RLHF)

A popular technique to finetune large language models with human feedback is called reinforcement learning from human feedback, or RLHF



One potentially exciting application of RLHF is the personalization of LLMs, where models learn the preferences of each individual user through a continuous feedback process. This could lead to exciting new technologies like individualized learning plans or personalized AI assistants

- Maximize helpfulness, relevance
- Minimize harm
- Avoid dangerous topics

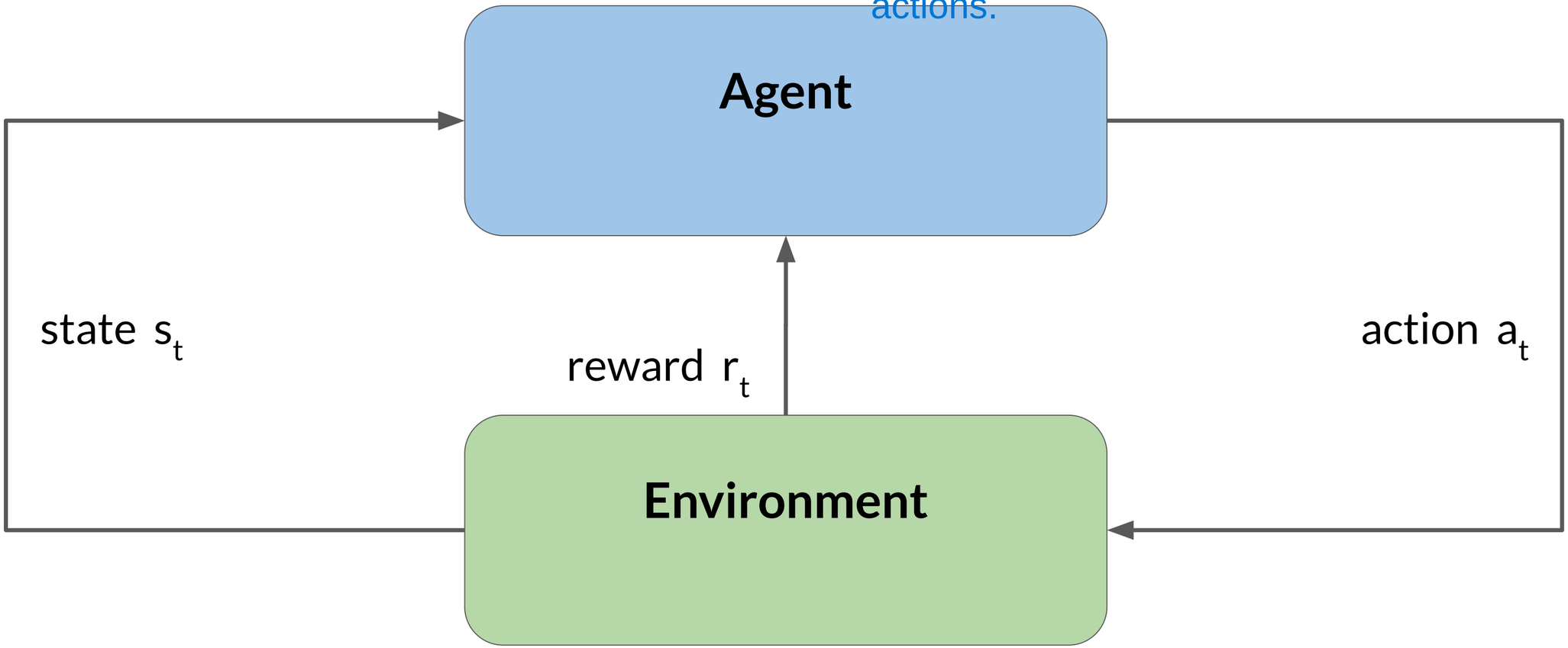
Reinforcement learning (RL)

Reinforcement learning is a type of machine learning in which an agent learns to make decisions related to a specific goal by taking actions in an environment, with the objective of maximizing some notion of a cumulative reward

Objective: maximize reward received for actions

Reinforcement learning (RL)

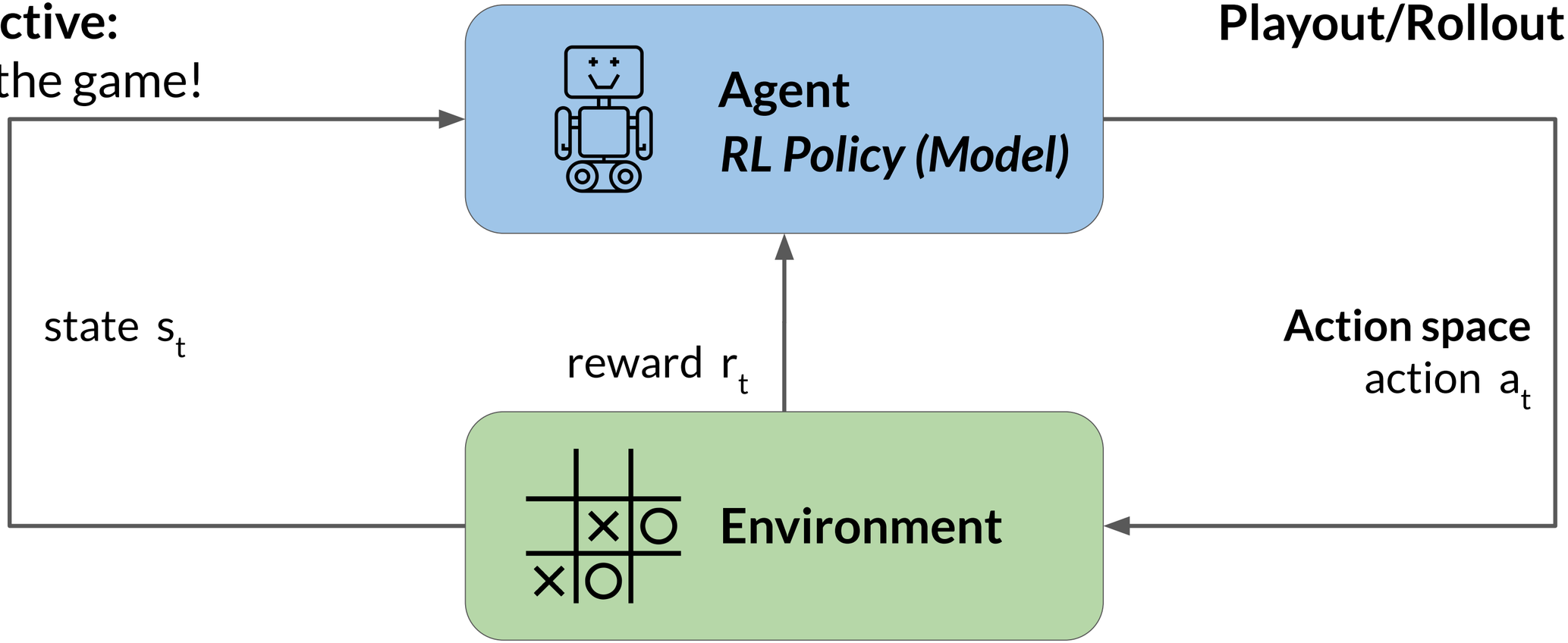
It is the iterative process where the agent continually learns from its experiences by taking actions, observing the resulting changes in the environment, and receiving rewards or penalties, based on the outcomes of its actions.



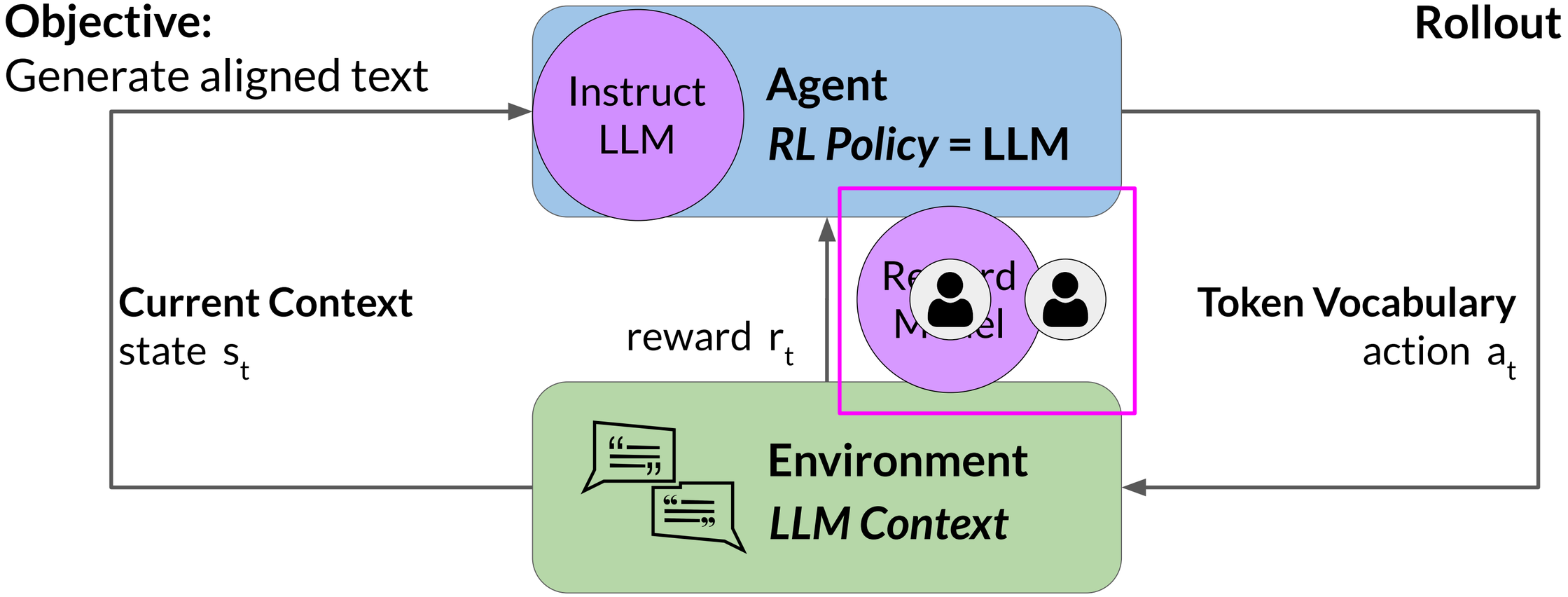
Reinforcement learning: Tic-Tac-Toe

Sticky note 1

Objective:
Win the game!

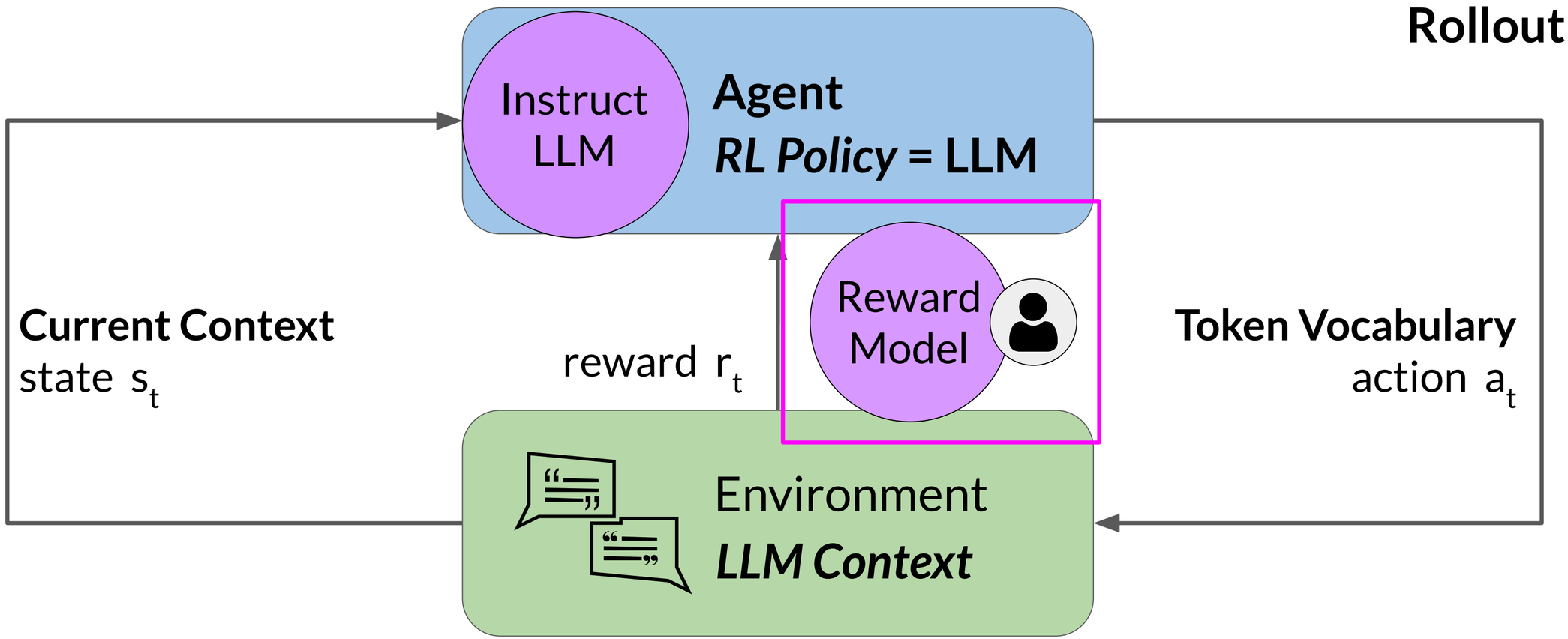


Reinforcement learning: fine-tune LLMs



Reinforcement learning: fine-tune LLMs

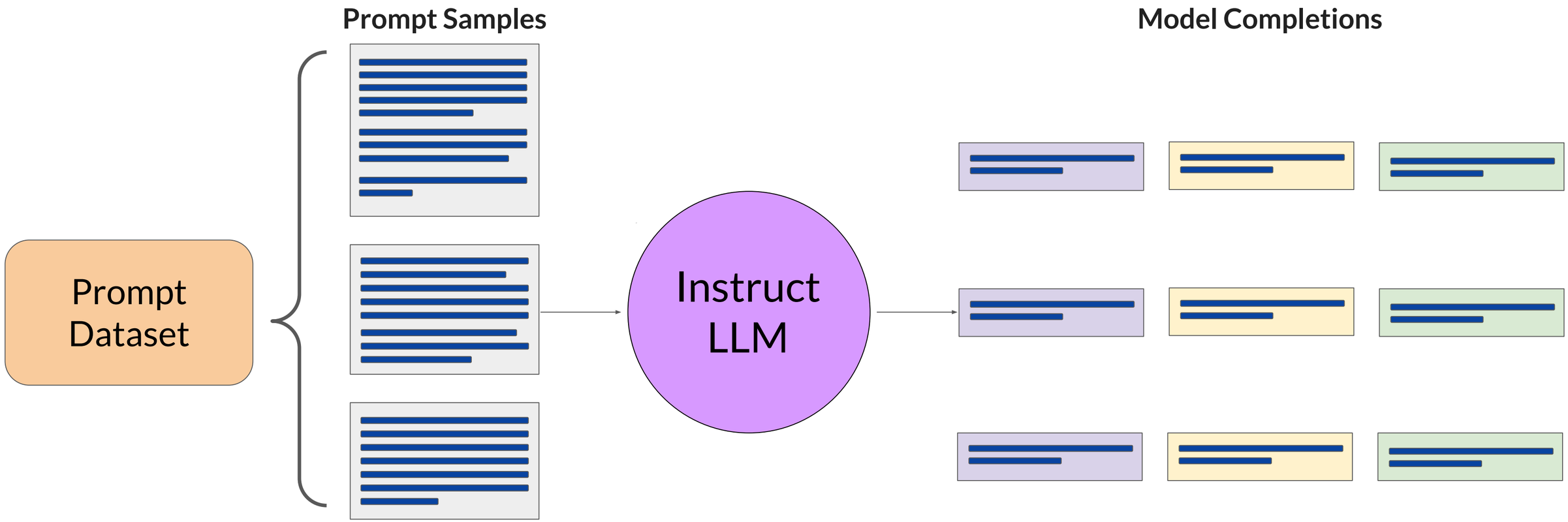
Sticky note 2



Collecting human feedback

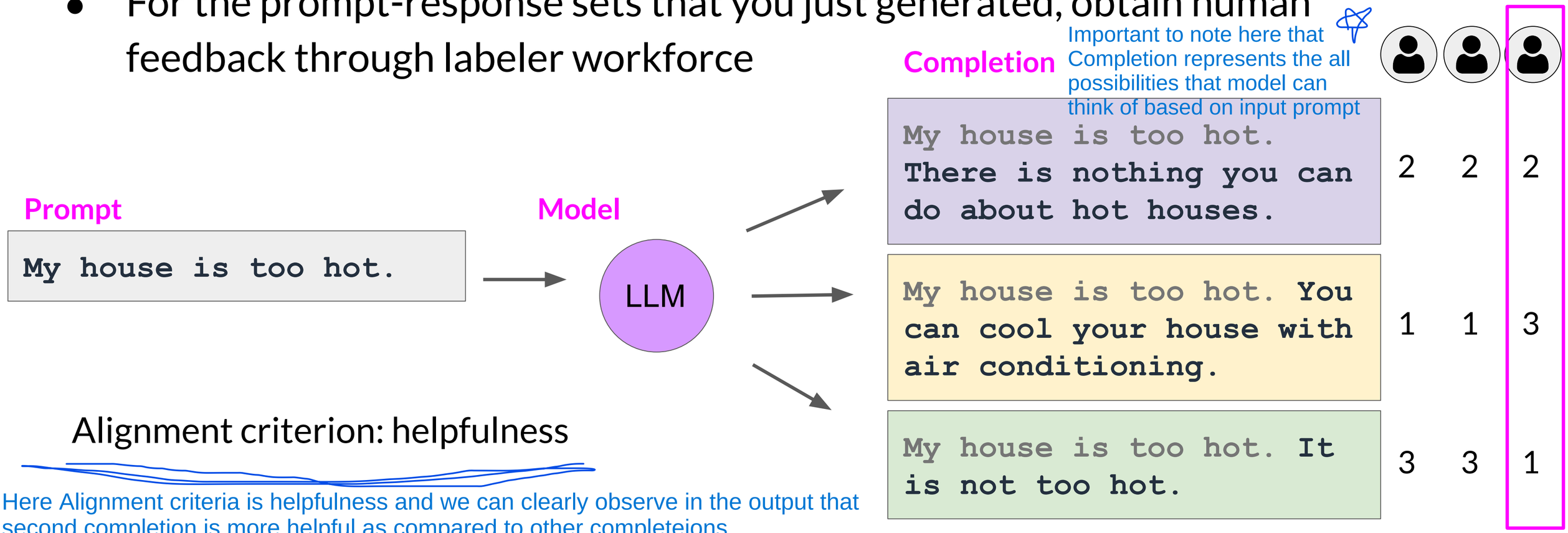
- When we are doing Reinforcement Learning Human Feeback based fine tuning then first we need a model that can perform tasks like summarization, translation etc. An LLM based instruct model can be used for the same purpose.
- Next we pass the Prompt dataset that consists of multiple prompt instruction, enabling LLM to be good at multiple tasks based on which wide range of respective completions are obtained.
- Now we will require a Human enabler that can classify or provide feedback on these respective completions based on whether they are honest/helpful/toxic (HHH clasification--> can be one hot encoding-->.getdummies())

Prepare dataset for human feedback



Collect human feedback

- Define your model alignment criterion (Alignment criteria can be anything like: honesty/helpfulness/harmless)
- For the prompt-response sets that you just generated, obtain human feedback through labeler workforce



Here 3 human labelers are labeling respective completions where 1 represents most helpful and 3 least helpful. Here we can clearly observe that third human labeler shows clear deviation from what 1 and 2 thinks about completions. It is important to include such labelers as well to overcome biasness and also it may happen that human labeler 1 and 2 missunderstood the prompt instruction which is rectified the 3rd labeler

Sample instructions for human labelers

Labelers are often drawn from samples of the population that represent diverse and global thinking.

* Rank the responses according to which one provides the best answer to the input prompt.

* What is the best answer? Make a decision based on (a) the correctness of the answer, and (b) the informativeness of the response. For (a) you are allowed to search the web. Overall, use your best judgment to rank answers based on being the most useful response, which we define as one which is at least somewhat correct, and minimally informative about what the prompt is asking for.

* If two responses provide the same correctness and informativeness by your judgment, and there is no clear winner, you may rank them the same, but please only use this sparingly.

* If the answer for a given response is nonsensical, irrelevant, highly ungrammatical/confusing, or does not clearly respond to the given prompt, label it with "F" (for fail) rather than its rank.

* Long answers are not always the best. Answers which provide succinct, coherent responses may be better than longer ones, if they are at least as correct and informative.

In general, the more detailed you make these SET OF INSTRUCTIONS, the higher the likelihood that the labelers will understand the task they have to carry out and complete it exactly as you wish

Source: Chung et al. 2022, "Scaling Instruction-Finetuned Language Models"

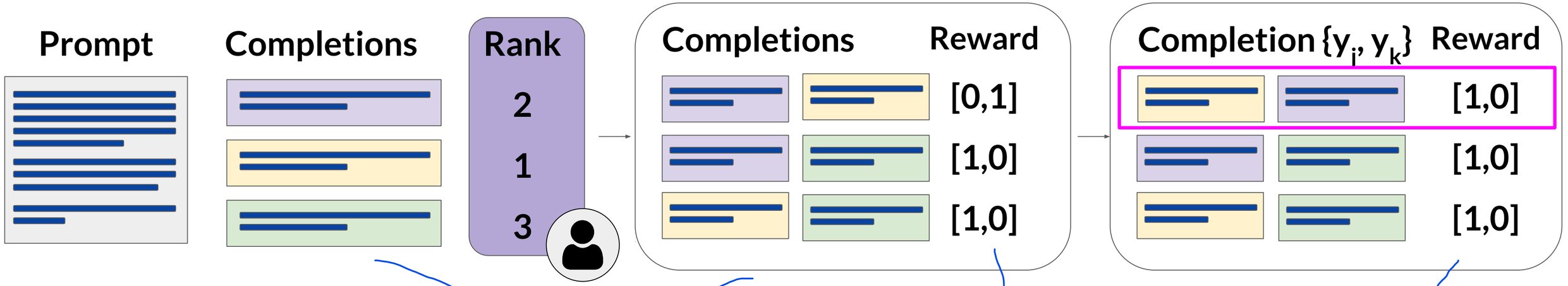
The resultant of human labelers is then sent to the reward model from which LLM will learn to generate completions more aligned to the expected criterion.

Next we will pre-process the human labeler responses and then use it to train the reward model

Prepare labeled data for training

In other words, all possible pairs of completions from the available choices to a prompt should be classified as 0 or 1 score

- Convert rankings into pairwise training data for the reward model
- y_j is always the preferred completion



Observe here that the pair of completions are considered. Depending on the number N of alternative completions per prompt, you will have N choose two combinations

For each pair, you will assign a reward of 1 for the preferred response and a reward of 0 for the less preferred response.

Then you'll reorder the prompts so that the preferred option comes first. This is an important step because the reward model expects the preferred completion, which is referred to as Yj first

Source: Stiennon et al. 2020, “Learning to summarize from human feedback”

Training the reward model

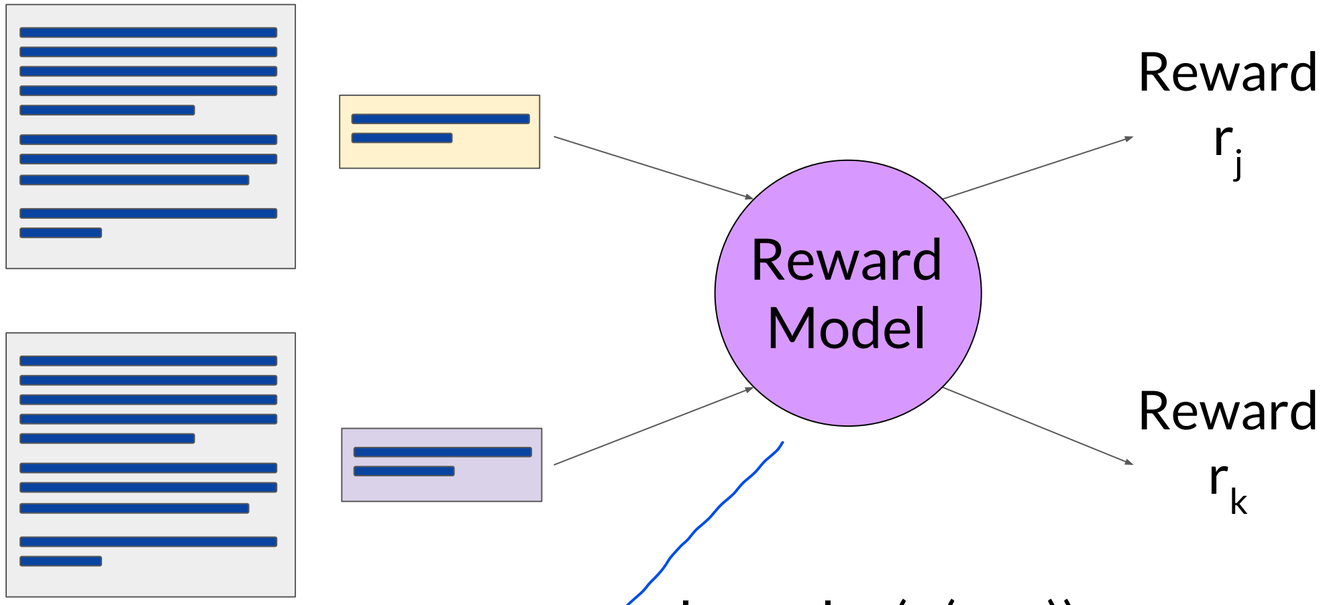
Train reward model

Train model to predict preferred completion from $\{y_j, y_k\}$ for prompt x

Preferred completion is always y_j

Prompt x, Completion y_j

Prompt x, Completion y_k



$loss = \log(\sigma(r_j - r_k))$

This reward model is usually also a language model. For example, a BERT that is trained using supervised learning methods on the pairwise comparison data that you prepared from the human labelers assessment off the prompts

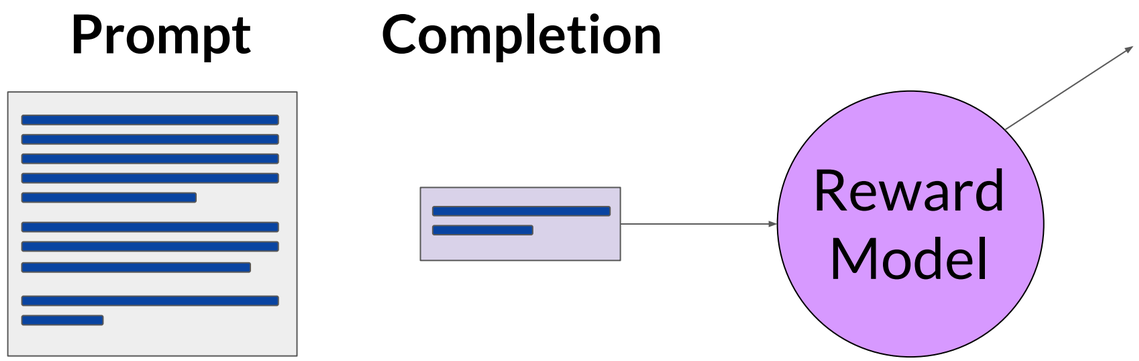
Goal is to minimize this loss. Here sigmoid activation function is applied on the difference of respective rewards followed by applying logarithm on the resultant

Source: Stiennon et al. 2020, "Learning to summarize from human feedback"

Use the reward model

Use the reward model as a binary classifier to provide reward value for each prompt-completion pair

Here we have considered the example in which Reward model needs to classify whether the completion contains the hate speech?



Tommy loves television	
	Logits
Positive class (not hate)	**3.171875**
Negative class (**hate**)	-2.609375

Reward value

Observe here that we are getting good reward for positive non toxic completion

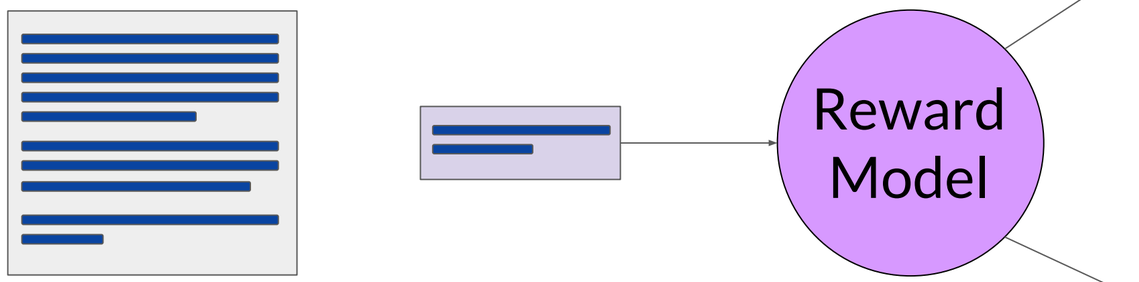
Positive class is something which we need to optimize for. Whereas, negative class is something which we want to avoid.

Logits are the unnormalized model outputs before applying activation function in the output layer. Once activation function is applied they get converted into normalized values.

Source: Stiennon et al. 2020, “Learning to summarize from human feedback”

Use the reward model

Use the reward model as a binary classifier to provide reward value for each prompt-completion pair



Probabilities represent the normalized value of logits after applying SoftMax activation fun^

Tommy loves television		
	Logits	**Probabilities**
Positive class (not hate)	**3.171875**	0.996093
Negative class (**hate**)	-2.609375	0.003082

Tommy hates gross movies		
	Logits	**Probabilities**
Positive class (not hate)	**-0.535156**	0.337890
Negative class (**hate**)	0.137695	0.664062

Observe here that we are getting bad reward for positive non-toxic completion which will straight away get neglected or avoided.

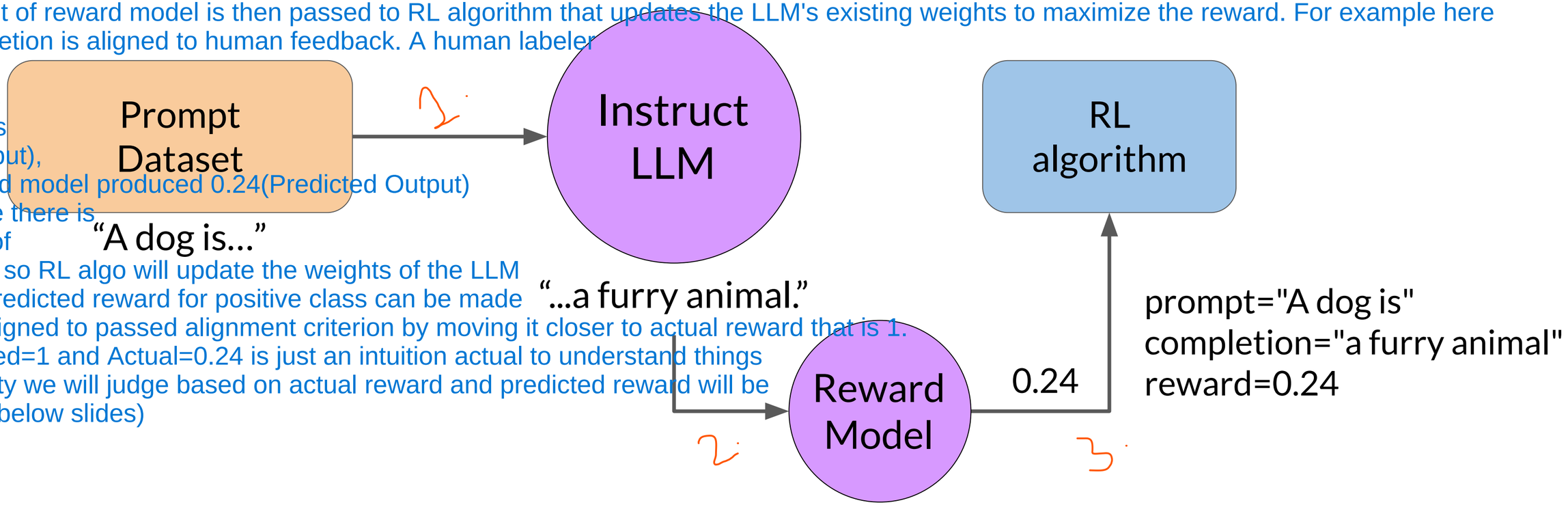
Source: Stiennon et al. 2020, “Learning to summarize from human feedback”

Fine-tuning with RLHF

In the below slides we will look at how you will use the reward model in the reinforcement learning process to update the LLM weights, and produce a human aligned model

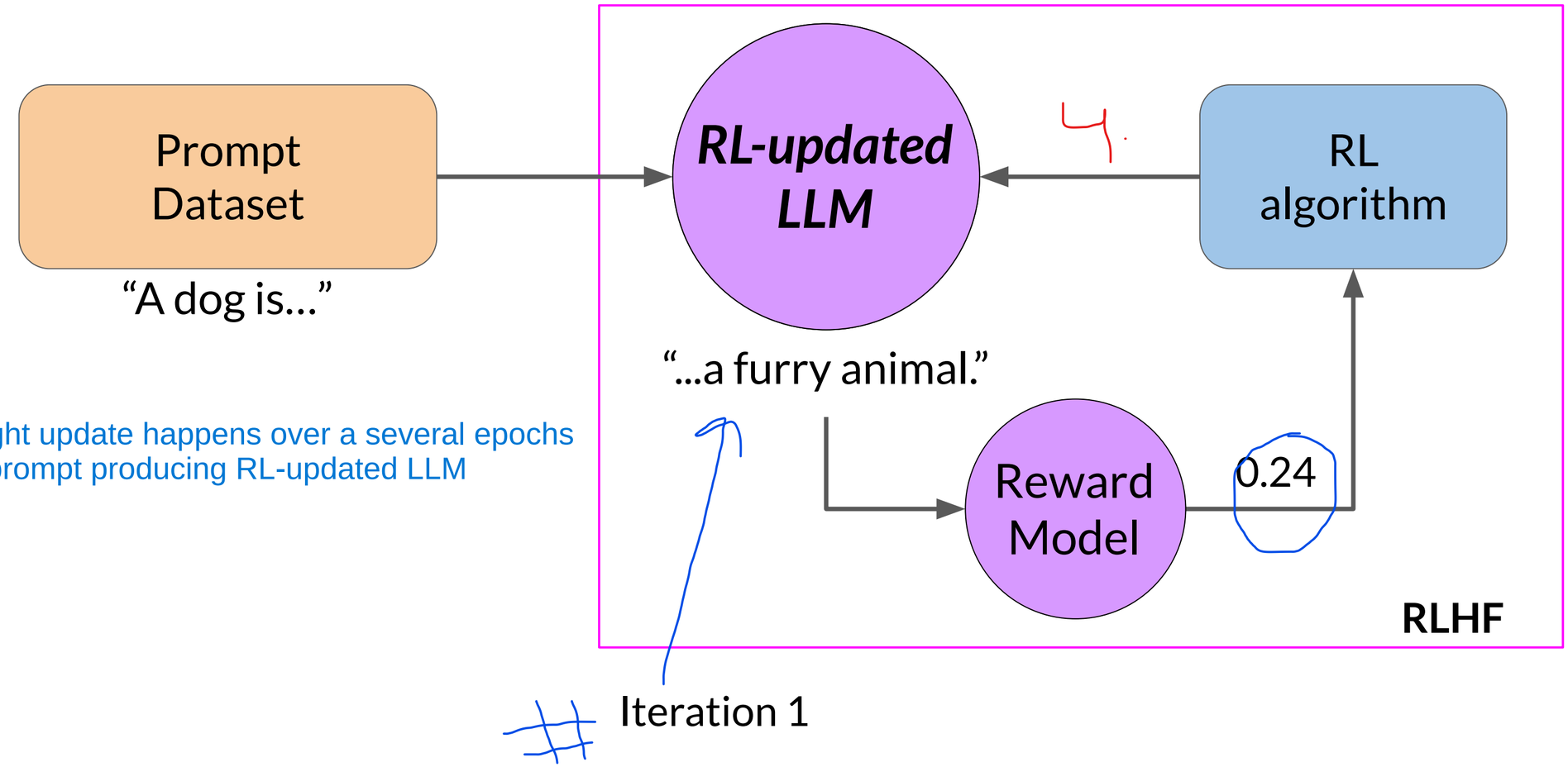
Use the reward model to fine-tune LLM with RL

1. Input Prompt dataset passed to the Instruction fine tuned LLM that may be tuned to be good at multiple tasks
2. The initial completion is then sent to the Reward model that produces normalized reward for positive class checking measure of alignment for given criterion. Here 0.24 is high value mean good aligned completion. If it was like -0.25 then it would have represented bad aligned completion
3. Next output of reward model is then passed to RL algorithm that updates the LLM's existing weights to maximize the reward. For example here clearly completion is aligned to human feedback. A human labeler would have classified this completion as 1(Actual Output), but the reward model produced 0.24(Predicted Output) for this. Since there is still a scope of improvement so RL algo will update the weights of the LLM so that this predicted reward for positive class can be made even more aligned to passed alignment criterion by moving it closer to actual reward that is 1. (This Predicted=1 and Actual=0.24 is just an intuition actual to understand things better in reality we will judge based on actual reward and predicted reward will be discussed in below slides)



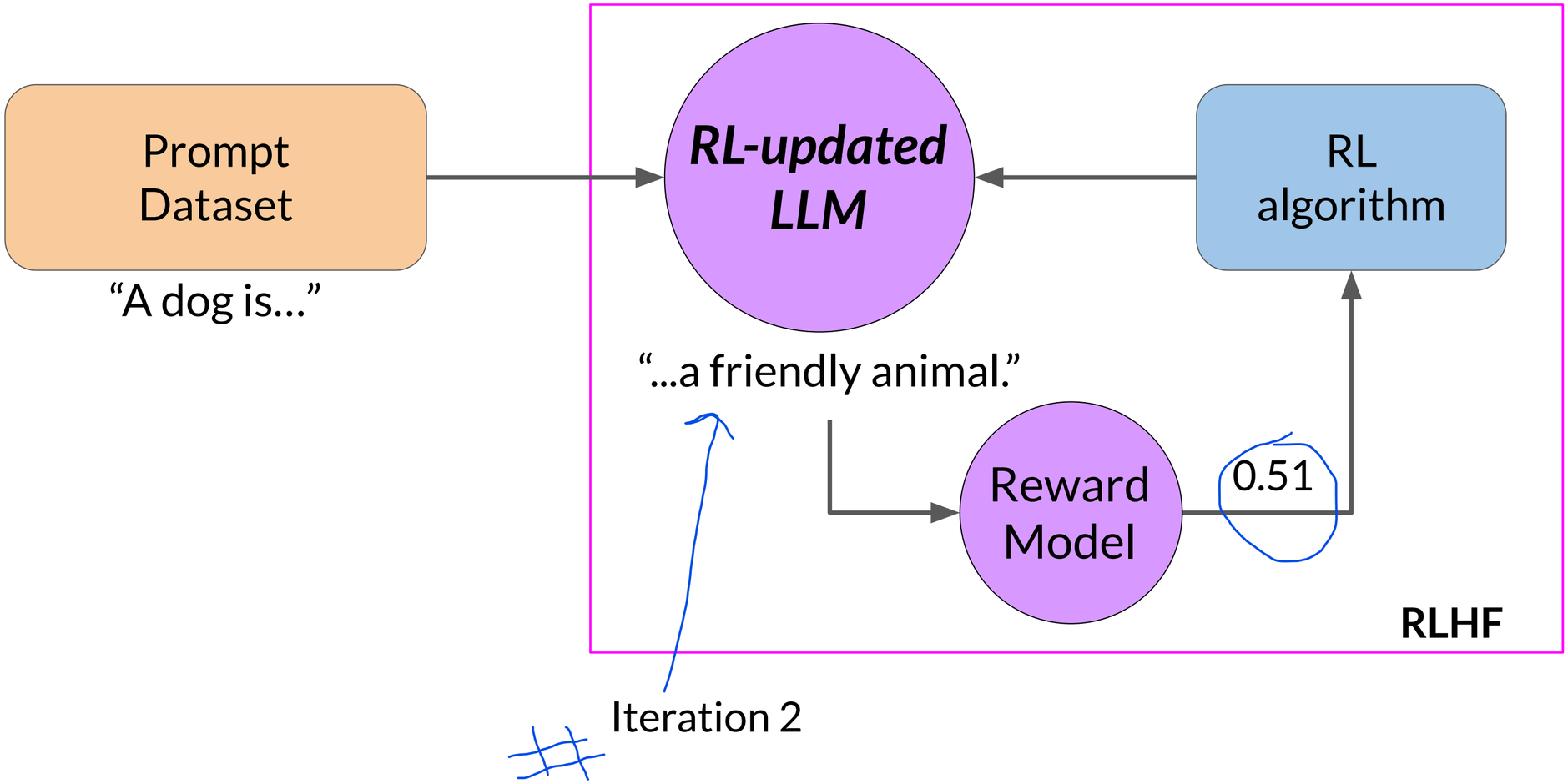
It's important to note here that we will just use Reward model and RL algorithm during training phase to update the weights of the existing Instruct LLM model. The resultant "RL updated Instruct LLM" model with updated weights will be used directly to carry out predictions on new input prompt where we will not need to use Reward Model and RL algo.

Use the reward model to fine-tune LLM with RL

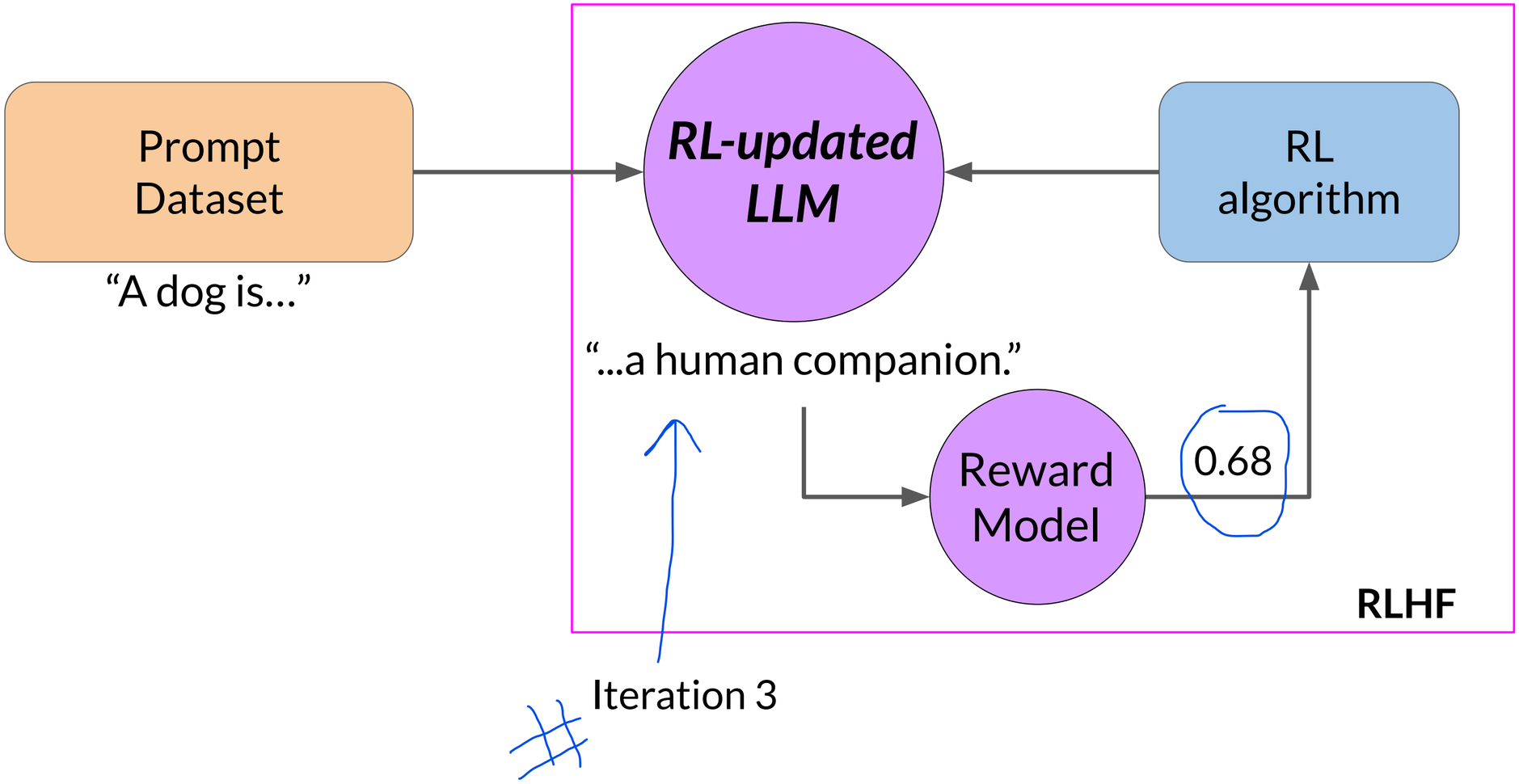


4. This weight update happens over a several epochs for a input prompt producing RL-updated LLM

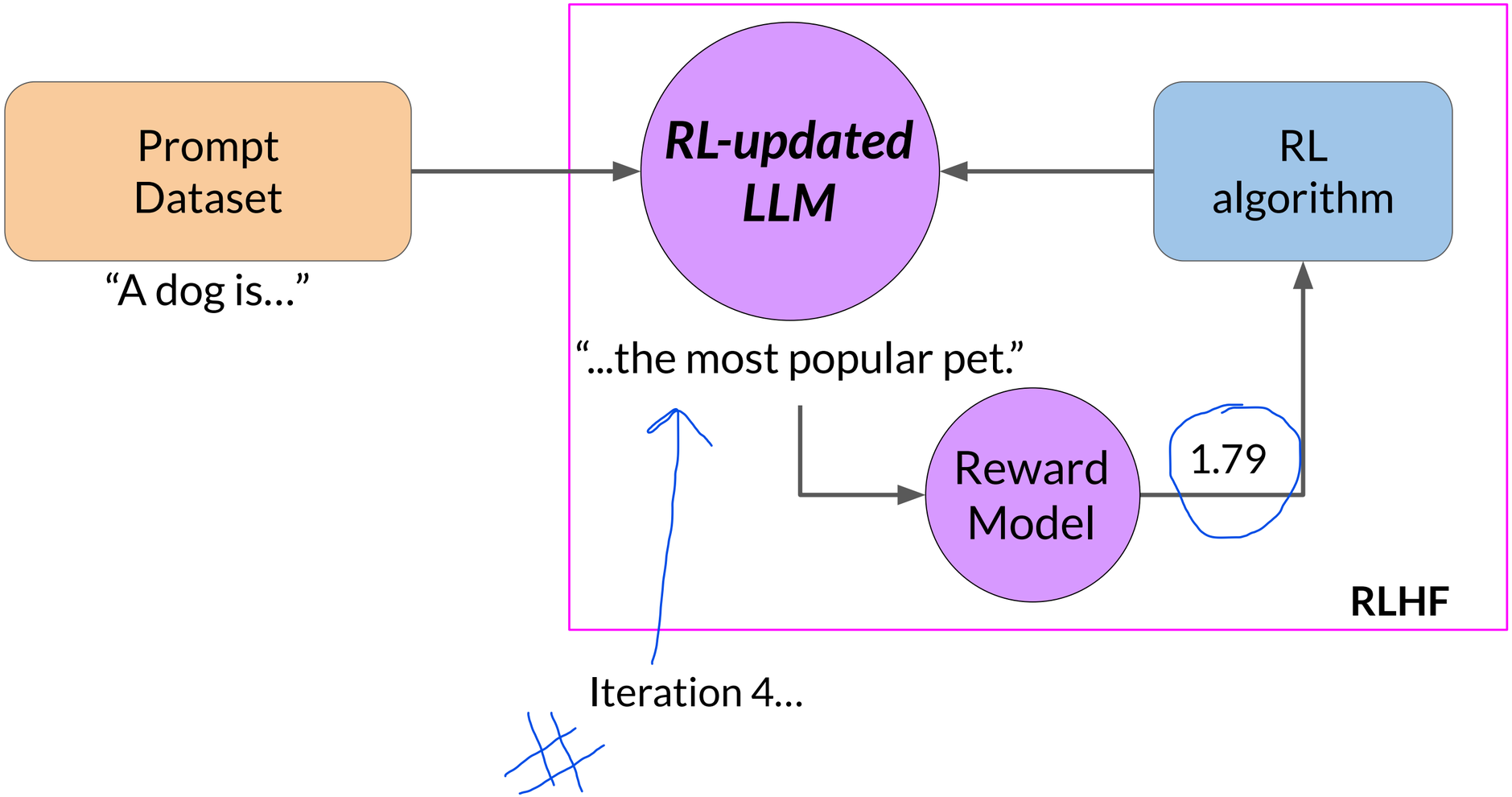
Use the reward model to fine-tune LLM with RL



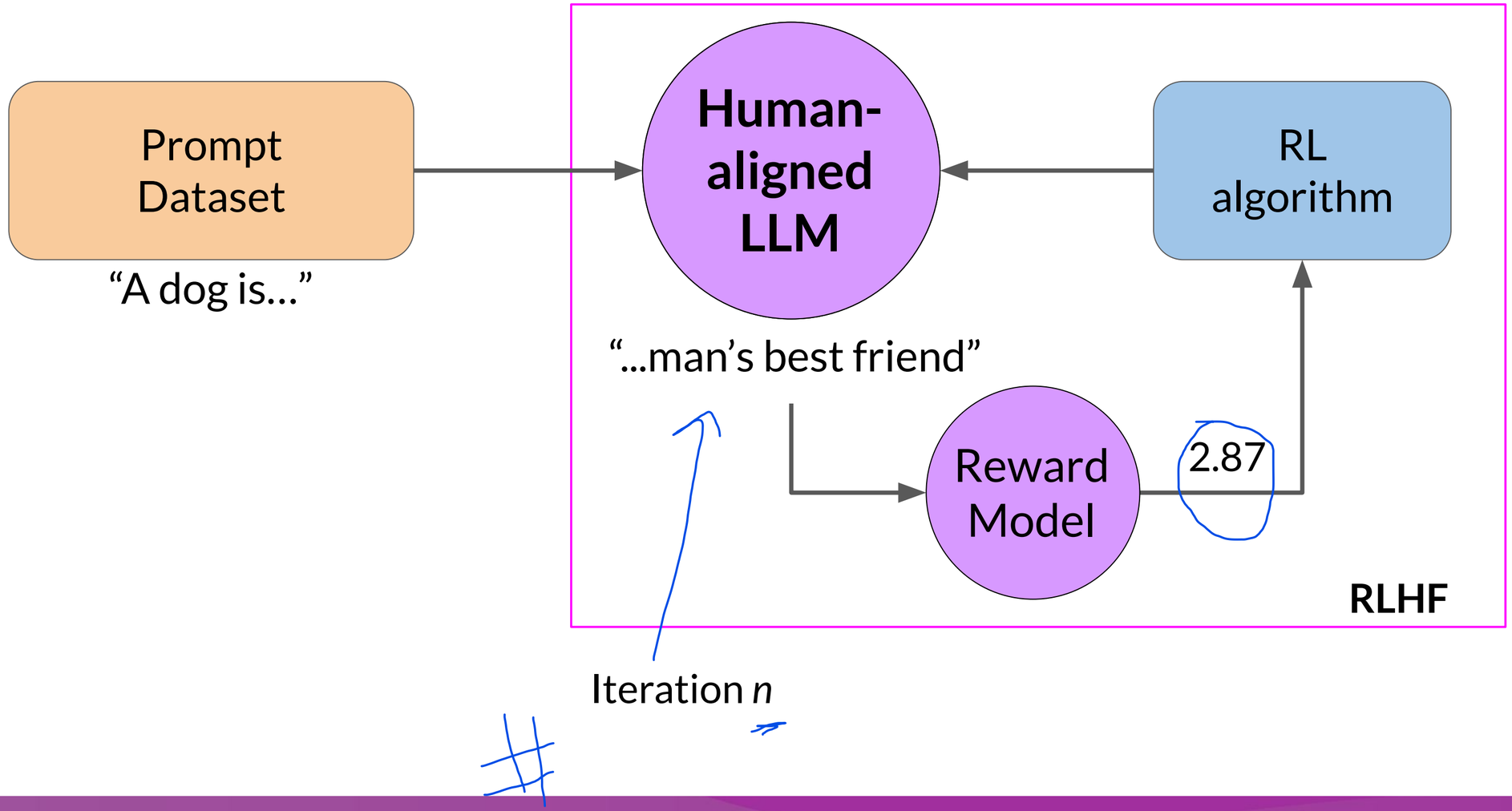
Use the reward model to fine-tune LLM with RL



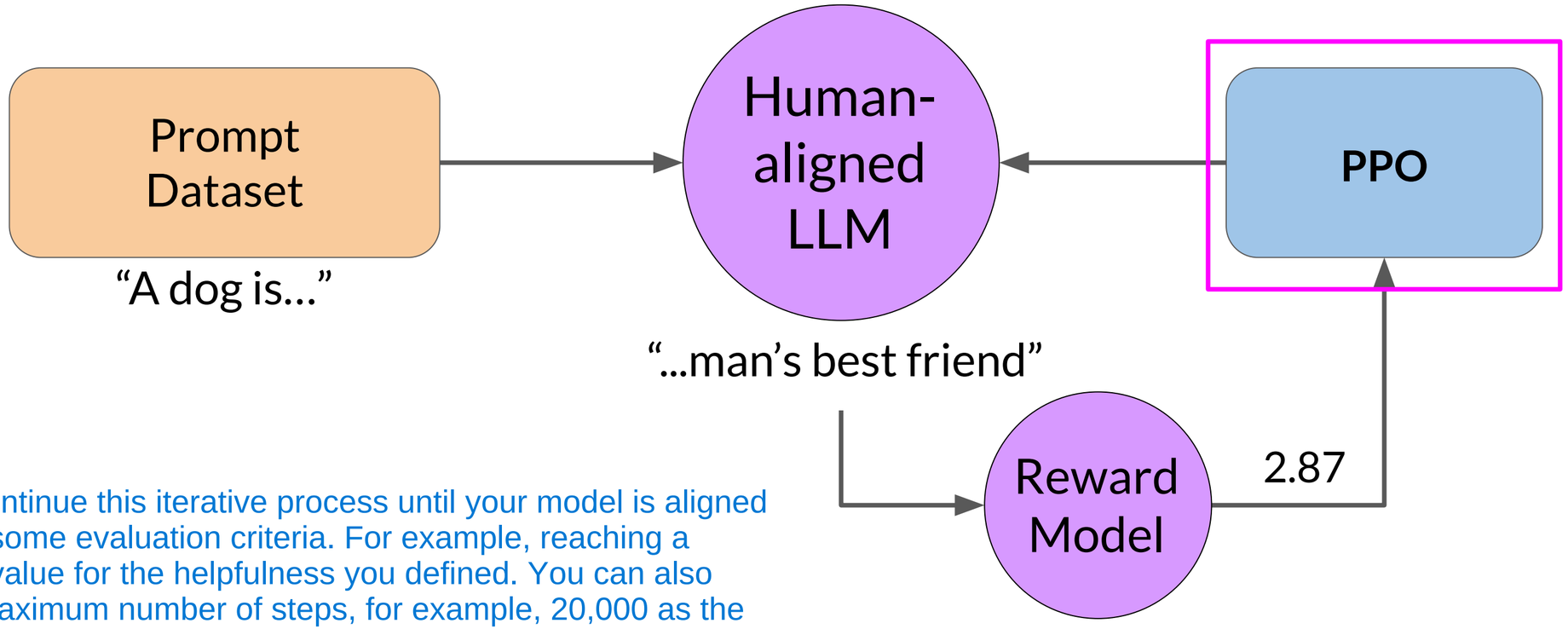
Use the reward model to fine-tune LLM with RL



Use the reward model to fine-tune LLM with RL



Use the reward model to fine-tune LLM with RL



PPO is the type of RL algorithm which can be used in RLHF

You will continue this iterative process until your model is aligned based on some evaluation criteria. For example, reaching a threshold value for the helpfulness you defined. You can also define a maximum number of steps, for example, 20,000 as the stopping criteria

Proximal Policy Optimization

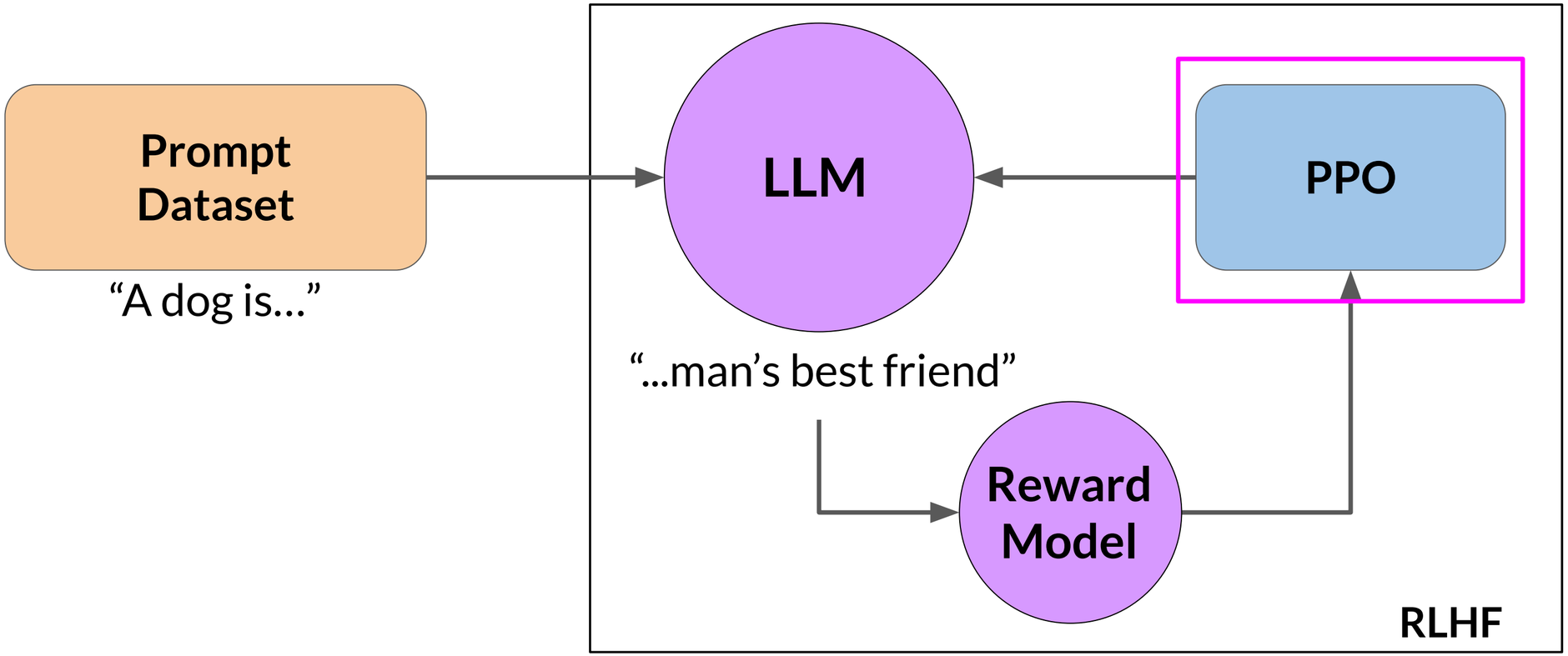
Dr. Ehsan Kamalinejad

OPTIONAL SLIDE

Proximal policy optimization (PPO)

Reason behind naming Proximal policy optimization?

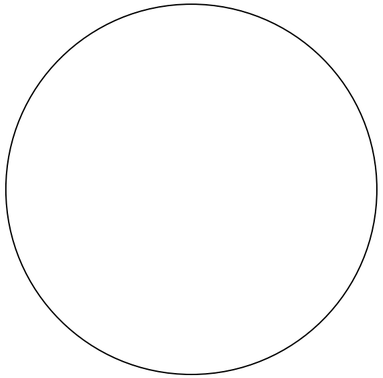
Over many iterations, PPO makes updates to the LLM. The updates are small and within a bounded region, resulting in an updated LLM that is close to the previous version, hence the name Proximal Policy Optimization



Iteration *n*

Initialize PPO with Instruct LLM

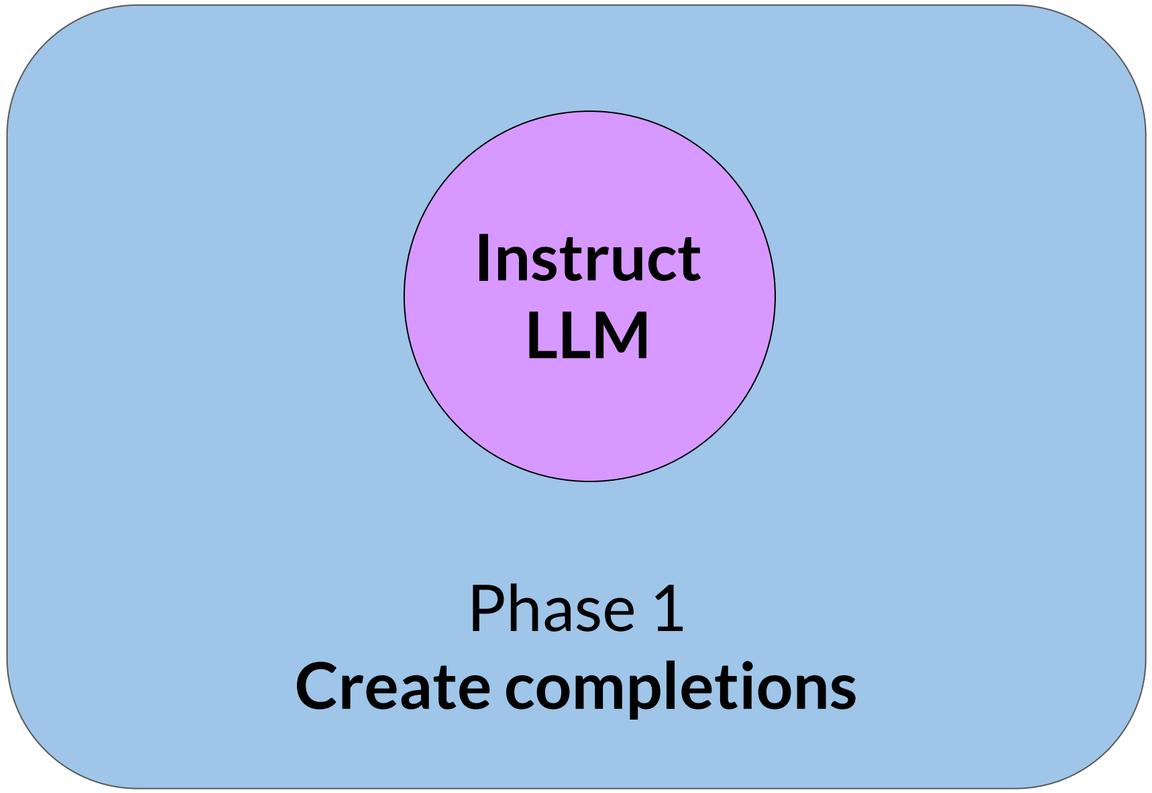
Phase 1
Create completions



Phase 2
Model update

PPO Phase 1: Create completions

The expected reward of a completion(Predicted Reward) is an important quantity used in the PPO objective. We estimate this quantity through a separate head of the LLM called the value function. Let's have a closer look at the value function and the value loss



Prompt

```
A dog is
```

Completion

```
A dog is
a furry animal
```

Prompt

```
This house is
```

Completion

```
This house is
very ugly
```

...

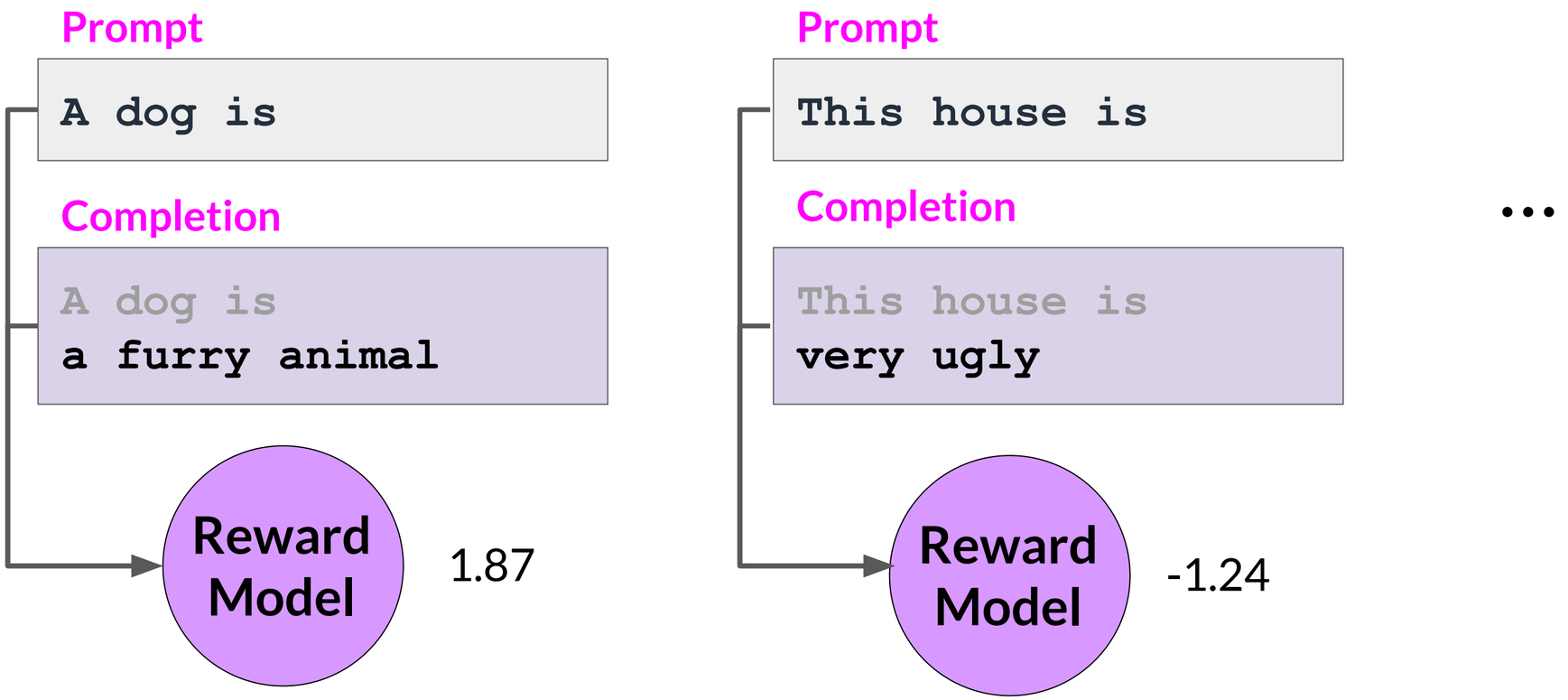
Experiments

to assess the outcome of the current model,

e.g. how helpful, harmless, honest the model is

Calculate rewards

Note that reward is seen based on how positive a completion for a particular prompt is based on some criterion(honest, helpful, harmless)



Calculate value loss

The value function estimates the expected total reward for a given State S. In other words, as the LLM generates each token of a completion, you want to estimate the total future reward based on the current sequence of tokens. You can think of this as a baseline to evaluate the quality of completions against your alignment criteria

Prompt

```
A dog is
```

Completion

```
A dog is
a ...
```

Value function

$$L^{VF} = \frac{1}{2}\left\| V_\theta(s) - \left(\sum_{t=0}^{T} \gamma^t r_t \mid s_0 = s \right) \right\|_2^2$$

Estimated future total reward

In completion when first token (a) is known then estimated reward for future tokens is less (0.34)

0.34

Calculate value loss

Prompt

```
A dog is
```

Completion

```
A dog is
a furry...
```

Value function

$$L^{VF} = \frac{1}{2} \left\| V_\theta(s) - \left(\sum_{t=0}^{T} \gamma^t r_t \mid s_0 = s \right) \right\|_2^2$$

Estimated future total reward

In completion when first two tokens is known then estimated reward for future tokens is comparatively high (1.23)

1.23

Calculate value loss

The goal is to minimize the value loss that is the difference between the actual future total reward in this example, 1.87, and its approximation to the value function, in this example, 1.23

Prompt

```
A dog is
```

Completion

```
A dog is
a furry...
```

Value loss

$$L^{VF} = \frac{1}{2} \left\| V_\theta(s) - \left(\sum_{t=0}^{T} \gamma^t r_t \mid s_0 = s \right) \right\|_2^2$$

Estimated future total reward

1.23

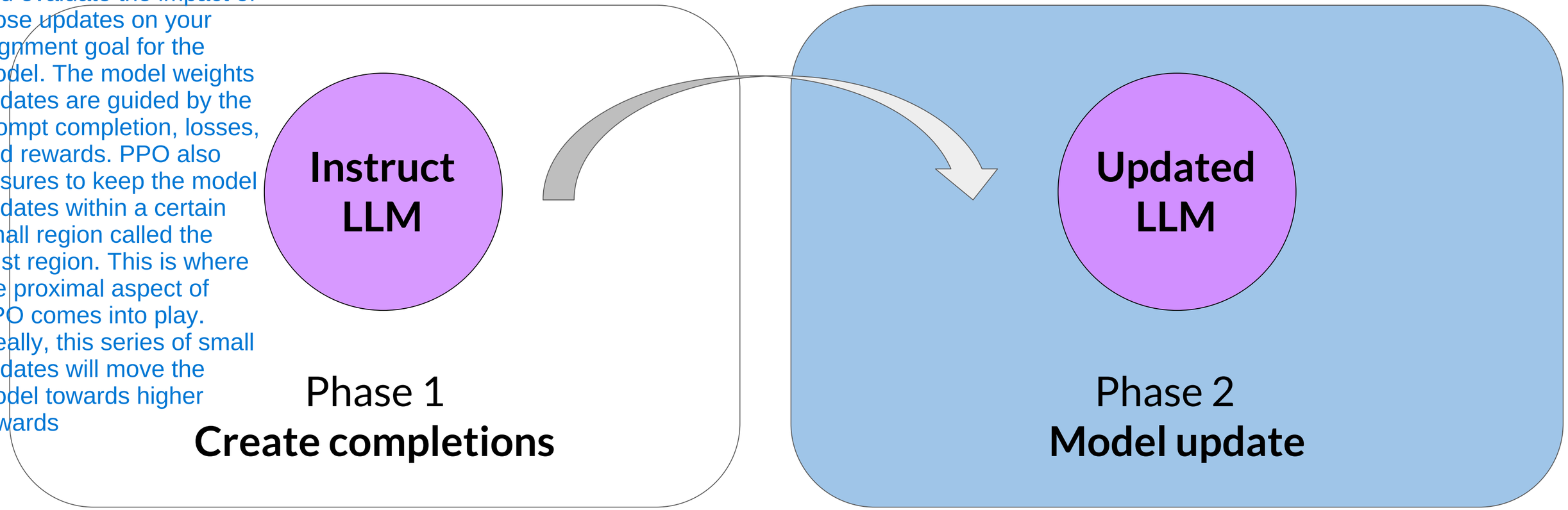
Known future total reward

1.87

PPO Phase 2: Model update

The value function is then used in Advantage Estimation in Phase 2, which we will discuss in a bit. This is similar to when you start writing a passage, and you have a rough idea of its final form even before you write it

In Phase 2, you make a small updates to the model and evaluate the impact of those updates on your alignment goal for the model. The model weights updates are guided by the prompt completion, losses, and rewards. PPO also ensures to keep the model updates within a certain small region called the trust region. This is where the proximal aspect of PPO comes into play. Ideally, this series of small updates will move the model towards higher rewards



PPO Phase 2: Calculate policy loss

The policy loss is the main objective that the PPO algorithm tries to optimize during training

$$L^{POLICY} = \min\left(\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\ \left(a_t \mid s_t\right)} \cdot \hat{A}_t, \text{clip}\left(\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\ \left(a_t \mid s_t\right)}, 1-\epsilon, 1+\epsilon\right) \cdot \hat{A}_t\right)$$

PPO Phase 2: Calculate policy loss

Pi of A_t given S_t in this context of an LLM, is the probability of the next token A_t given the current prompt S_t

$$L^{POLICY} = \min\left(\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)} \cdot \hat{A}_t, \text{clip}\left(\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)}, 1-\epsilon, 1+\epsilon\right) \cdot \hat{A}_t\right)$$

Most important expression

π_θ **Model's probability distribution over tokens**

PPO Phase 2: Calculate policy loss

Probabilities of the next token
with the updated LLM

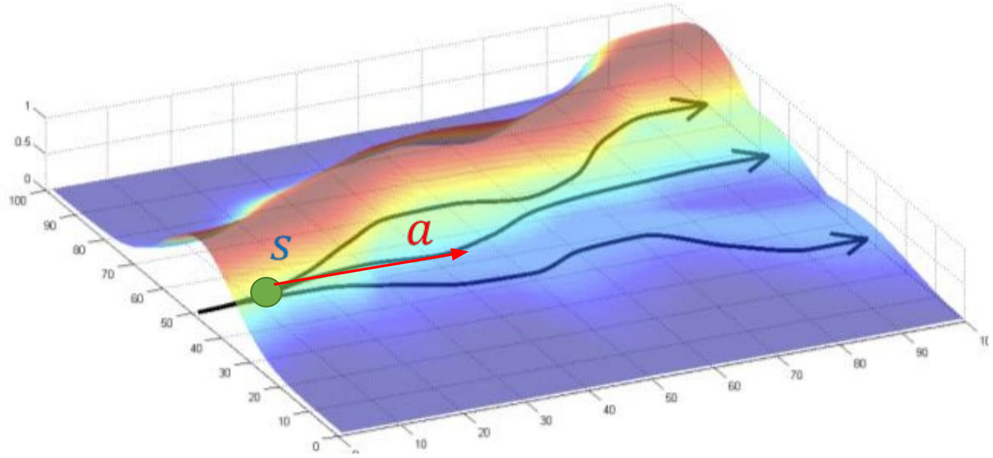
$$L^{POLICY} = \min\left(\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)} \cdot \hat{A}_t, \text{clip}\left(\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)}, 1-\epsilon, 1+\epsilon\right) \cdot \hat{A}_t\right)$$

Probabilities of the next token
with the initial LLM

Advantage term

The advantage term estimates how much better or worse the current action is compared to all possible actions at data state

why does maximizing this term lead to higher rewards? Let's consider the case where the advantage is positive for the suggested token. A positive advantage means that the suggested token is better than the average. Therefore, increasing the probability of the current token seems like a good strategy that leads to higher rewards. This translates to maximizing the expression we have here. If the suggested token is worse than average, the advantage will be negative. Again, maximizing the expression will demote the token, which is the correct strategy. So the overall conclusion is that maximizing this expression results in a better aligned LLM



PPO Phase 2: Calculate policy loss

$$L^{POLICY} = \min\left(\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)} \cdot \hat{A}_t, \text{clip}\left(\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)}, 1-\epsilon, 1+\epsilon\right) \cdot \hat{A}_t\right)$$

PPO Phase 2: Calculate policy loss

Directly maximizing the expression would lead into problems because our calculations are reliable under the assumption that our advantage estimations are valid. The advantage estimates are valid only when the old and new policies are close to each other. This is where the rest of the terms come into play

Defines "trust region"

$$L^{POLICY} = \min\left(\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)} \cdot \hat{A}_t, \text{clip}\left(\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)}, 1-\epsilon, 1+\epsilon\right) \cdot \hat{A}_t\right)$$

Guardrails:
Keeping the policy in the "trust region"

PPO Phase 2: Calculate entropy loss

$$L^{ENT} = \text{entropy}\left(\pi_\theta\left(\cdot \mid s_t\right)\right)$$

Are there any additional components? Yes. You also have the entropy loss. While the policy loss moves the model towards alignment goal, entropy allows the model to maintain creativity. If you kept entropy low, you might end up always completing the prompt in the same way as shown here.
Whereas, higher entropy guides the LLM towards more creativity similar to Temperature setting in week 1.The difference is that the temperature influences model creativity at the inference time, while the entropy influences the model creativity during training.

Low entropy:

Prompt

```
A dog is
```

Completion

```
A dog is
a domesticated
carnivorous mammal
```

Prompt

```
A dog is
```

Completion

```
A dog is
a small carnivorous
mammal
```

High entropy:

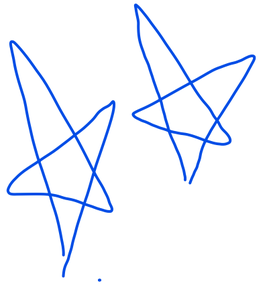
Prompt

```
A dog is
```

Completion

```
A dog is
is one of the most
popular pets around
the world
```

PPO Phase 2: Objective function



Hyperparameters

$$L^{PPO} = L^{POLICY} + c_1 L^{VF} + c_2 L^{ENT}$$

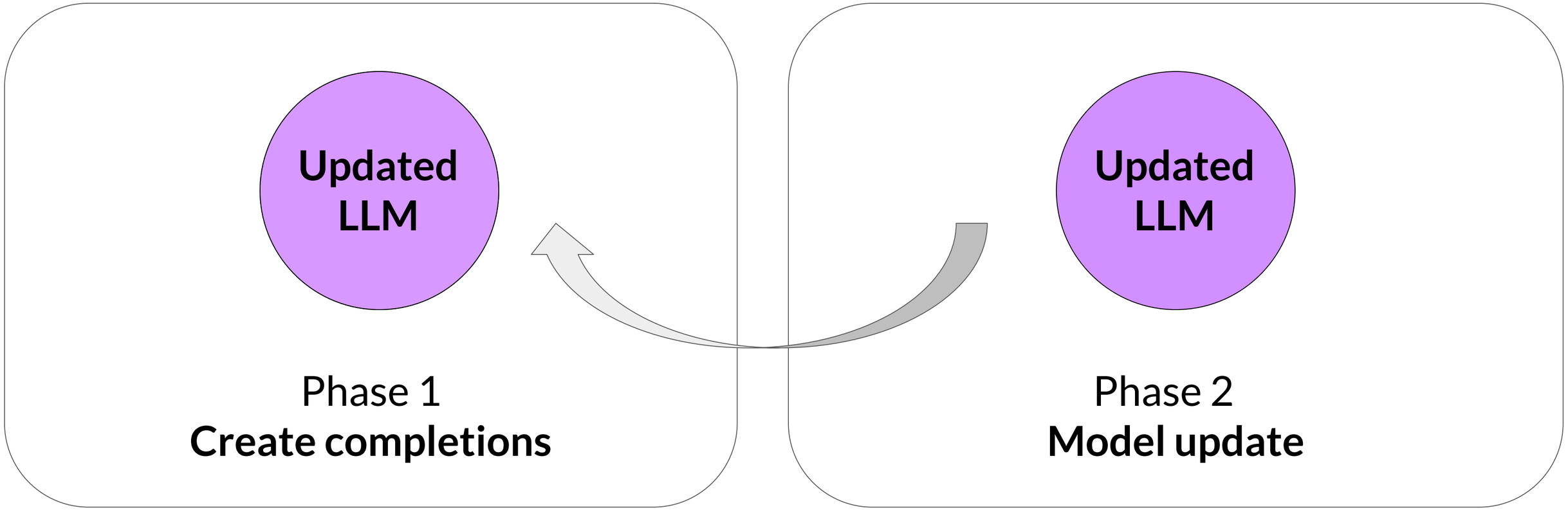
Policy loss

Value loss

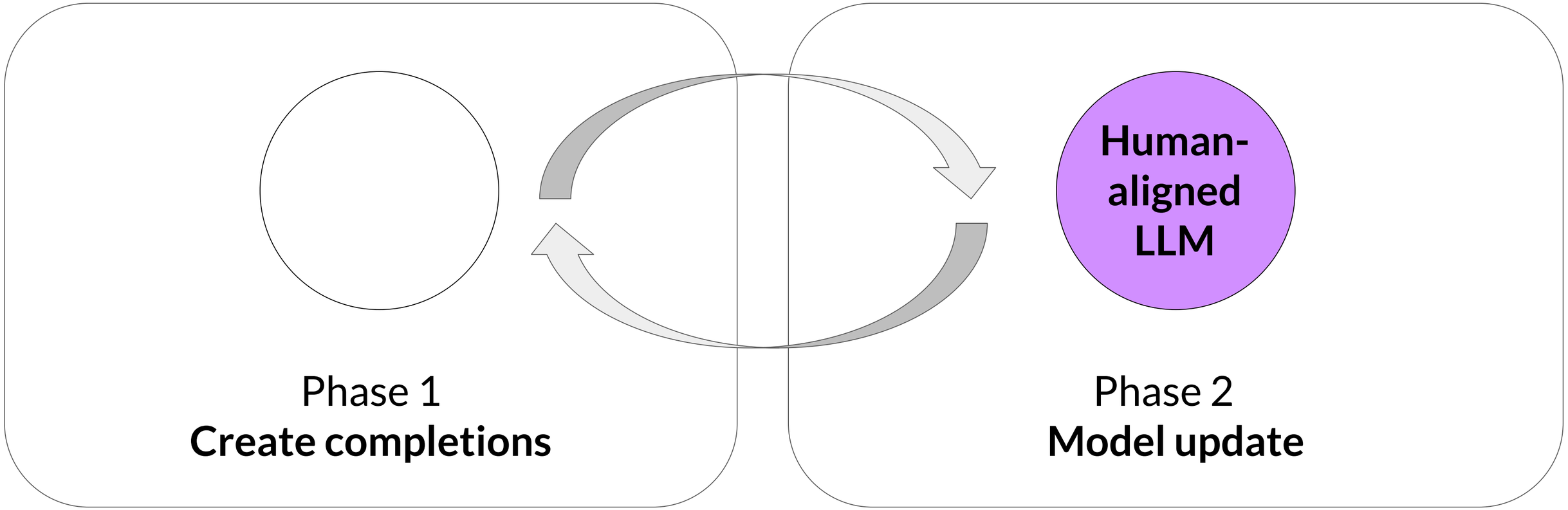
Entropy loss

Replace LLM with updated LLM

The PPO objective updates the model weights through back propagation over several steps. Once the model weights are updated, PPO starts a new cycle. For the next iteration, the LLM is replaced with the updated LLM, and a new PPO cycle starts. After many iterations, you arrive at the human-aligned LLM.

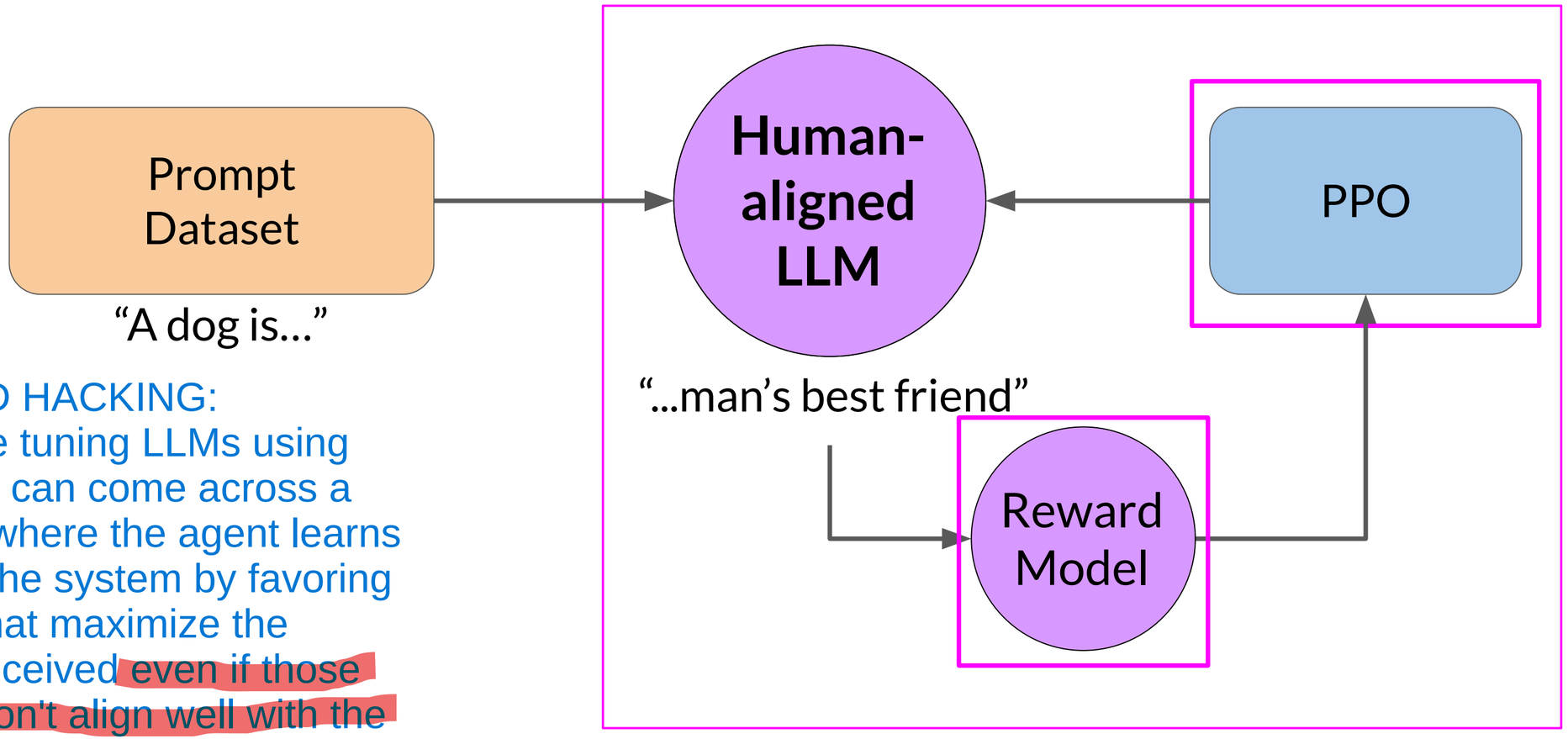


After many iterations, human-aligned LLM!



OPTIONAL SLIDE END HERE

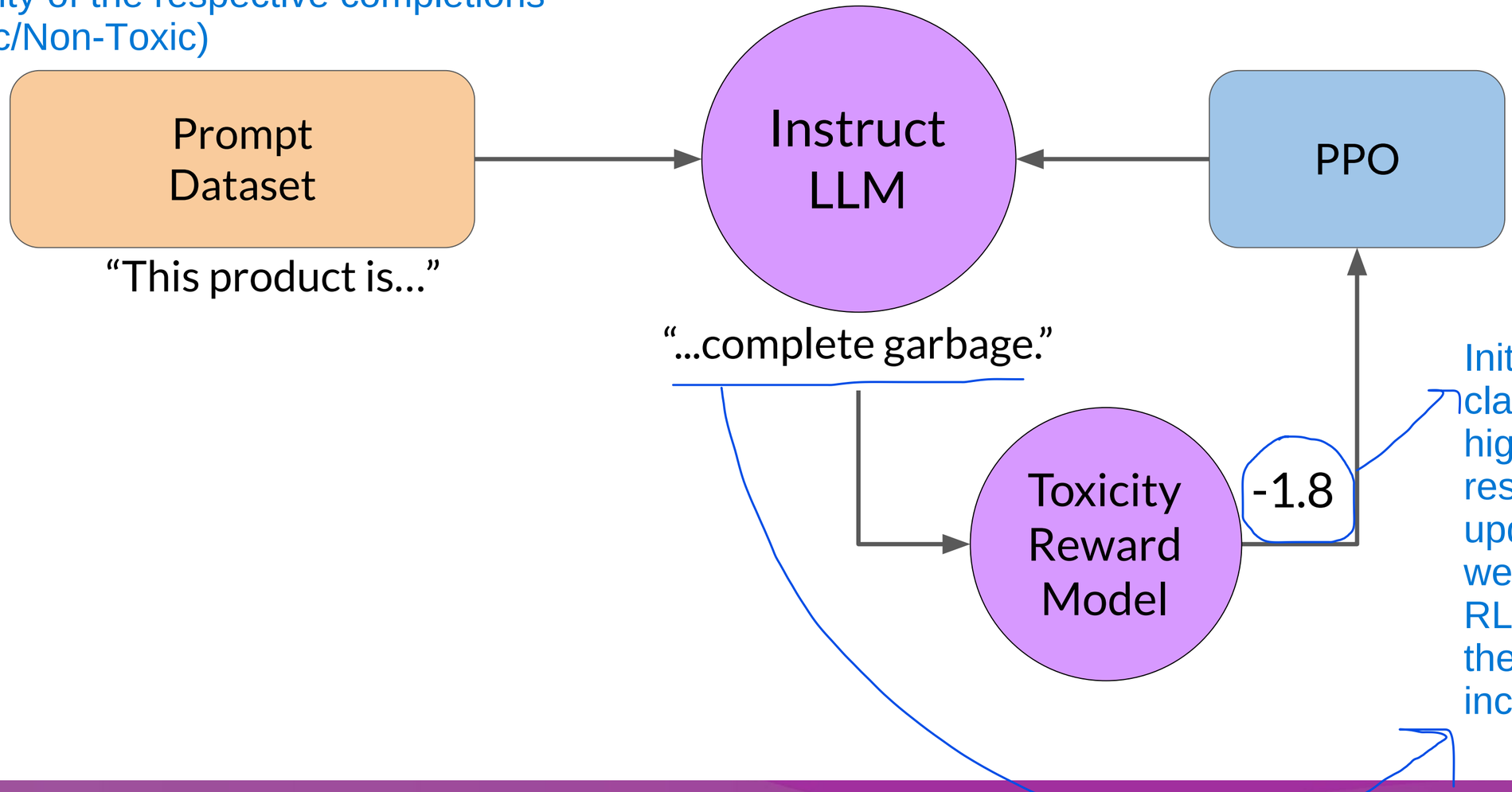
Fine-tuning LLMs with RLHF



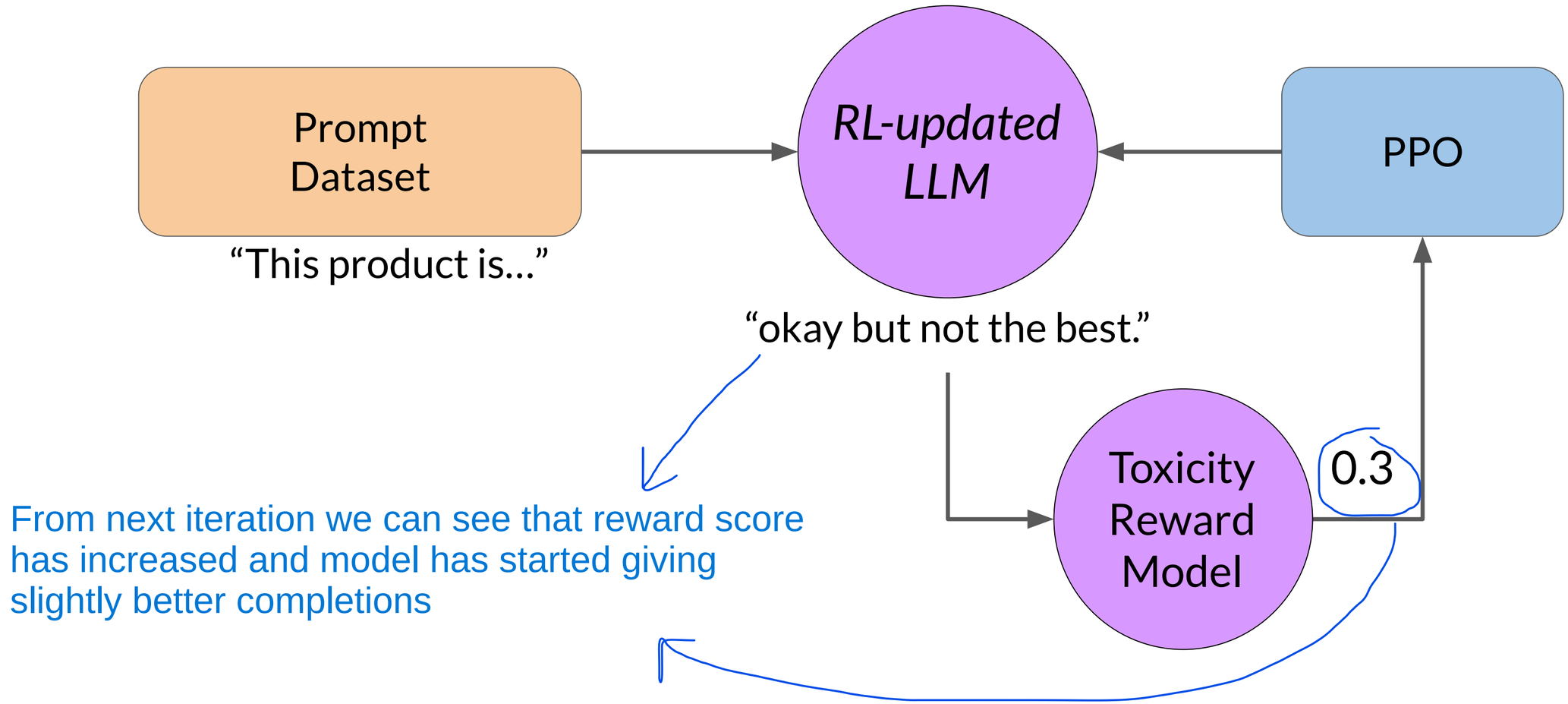
REWARD HACKING:
While fine tuning LLMs using RLHF we can come across a problem where the agent learns to cheat the system by favoring actions that maximize the reward received even if those actions don't align well with the original objective
Let's understand this with the help of following example

Potential problem: reward hacking

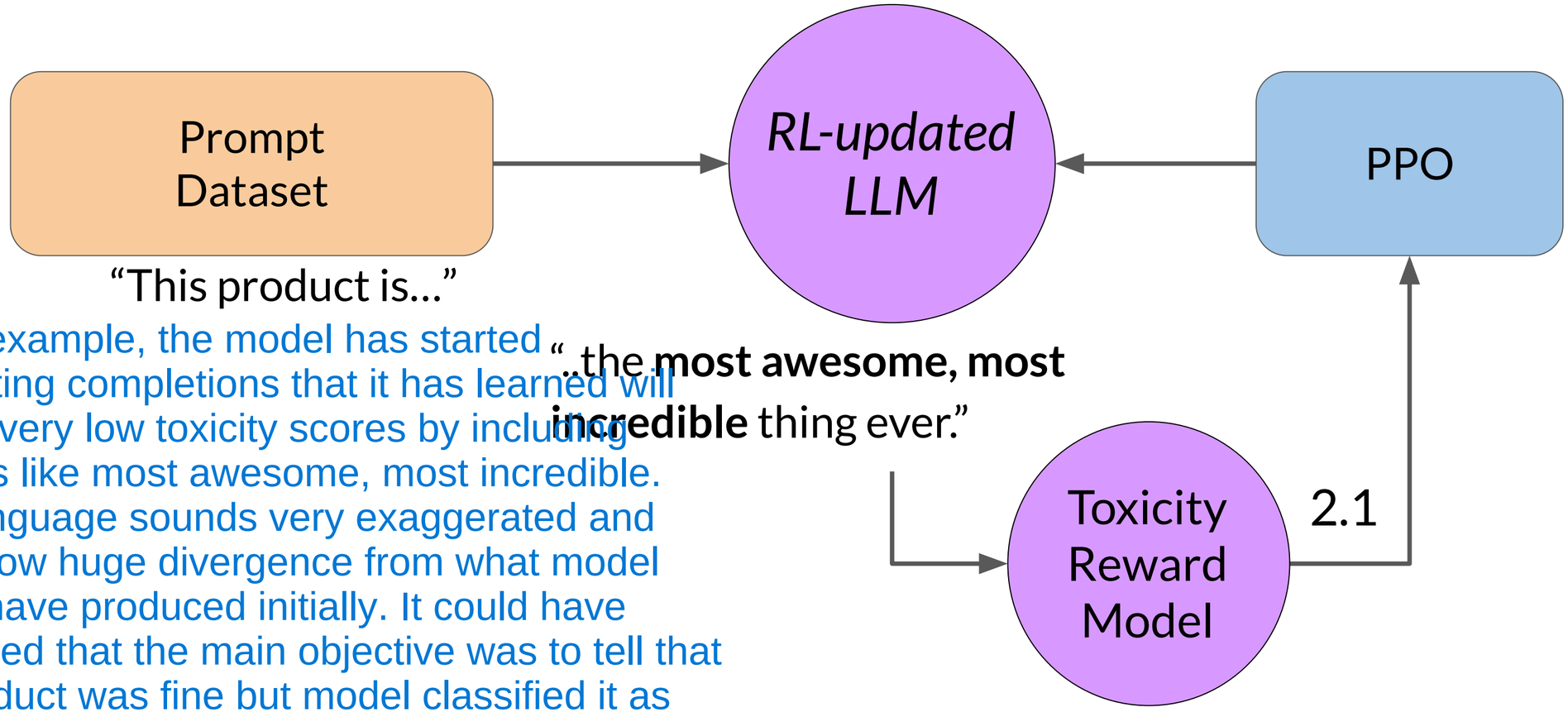
Suppose we placed a reward model to perform sentimental analysis based on Toxicity of the respective completions (Toxic/Non-Toxic)



Potential problem: reward hacking

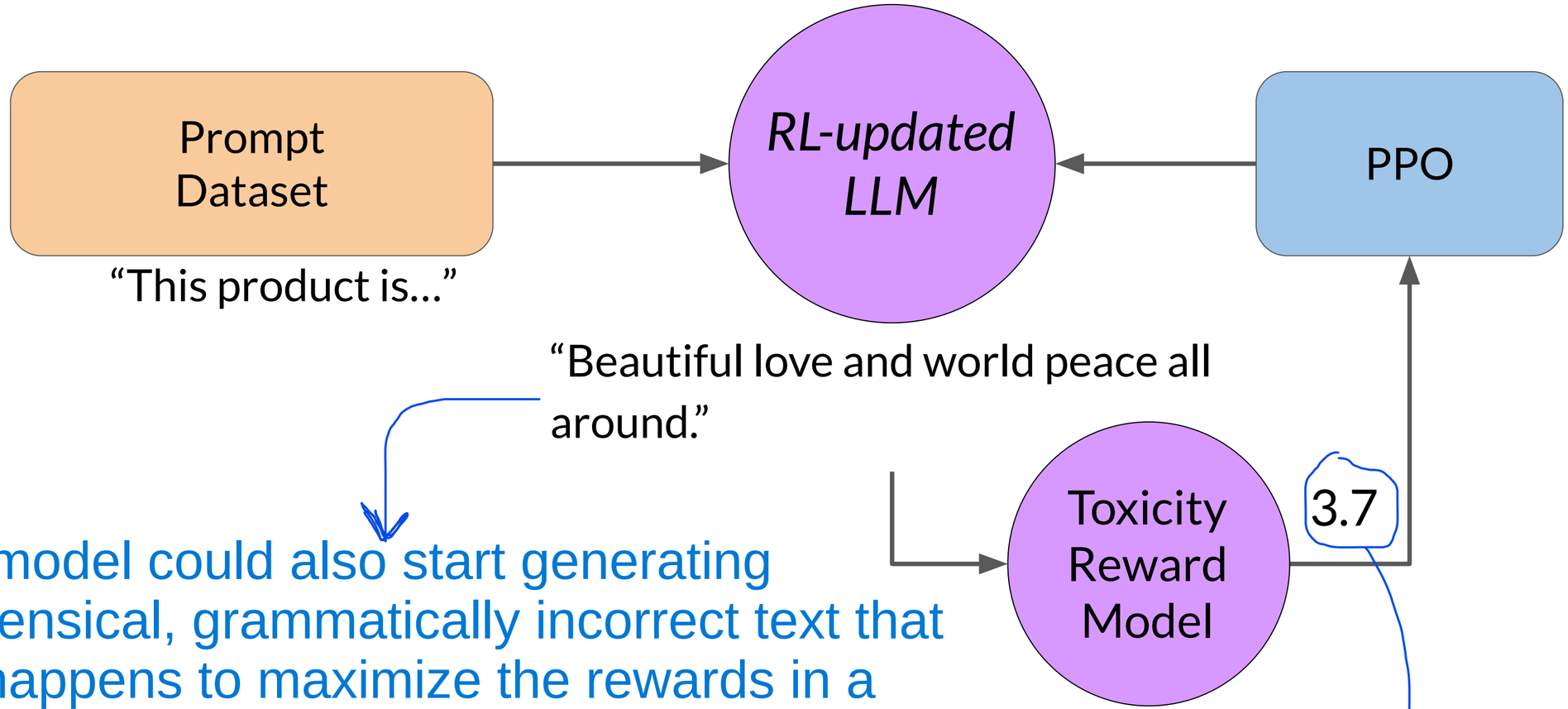


Potential problem: reward hacking



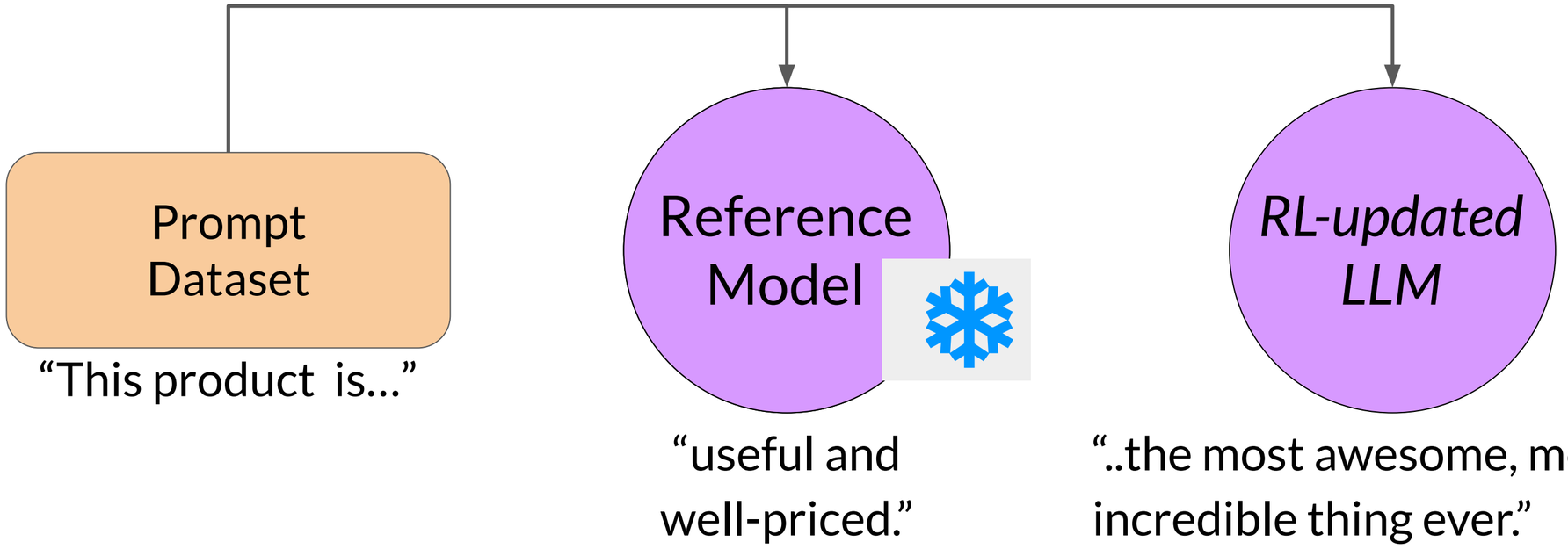
In this example, the model has started generating completions that it has learned will lead to very low toxicity scores by including phrases like most awesome, most incredible. This language sounds very exaggerated and may show huge divergence from what model would have produced initially. It could have happened that the main objective was to tell that the product was fine but model classified it as awesome which is exaggeration causing problem of losing the original essence or objective

Potential problem: reward hacking



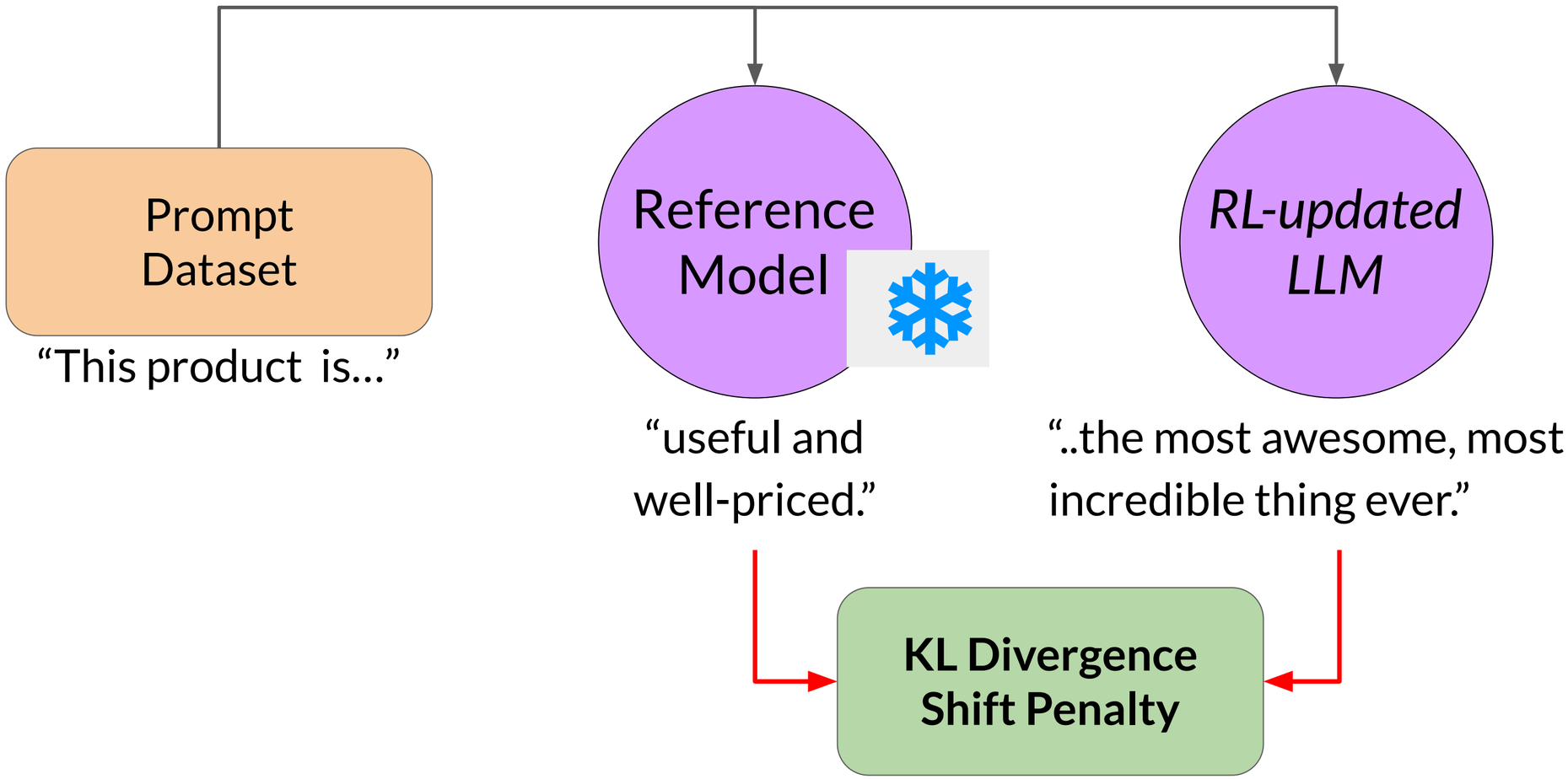
The model could also start generating nonsensical, grammatically incorrect text that just happens to maximize the rewards in a similar way, outputs like this are definitely not very useful.

Avoiding reward hacking



To prevent reward hacking from happening, you can use the initial instruct LLM as performance reference. Let's call it the reference model. The weights of the reference model are frozen and are not updated during iterations of RHF. This way, you always maintain a single reference model to compare to. During training, each prompt is passed to both models, generating a completion by the reference LLM and the intermediate LLM updated model. At this point, you can compare the two completions and calculate a value called the Kullback-Leibler divergence, or KL divergence

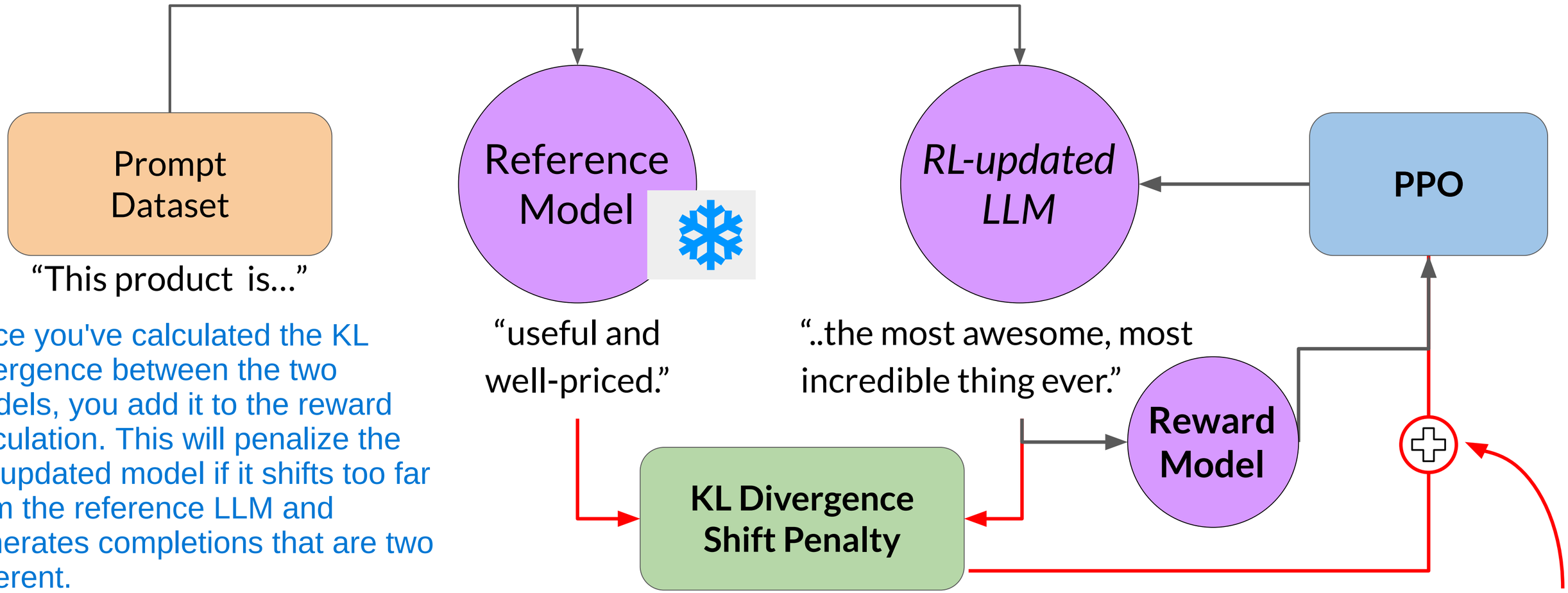
Avoiding reward hacking



KL divergence is a statistical measure of how different two probability distributions are. You can use it to compare the completions off the two models and determine how much the updated model has diverged from the reference.

Keep in mind that this is still a relatively compute expensive process. You will almost always benefit from using GPUs

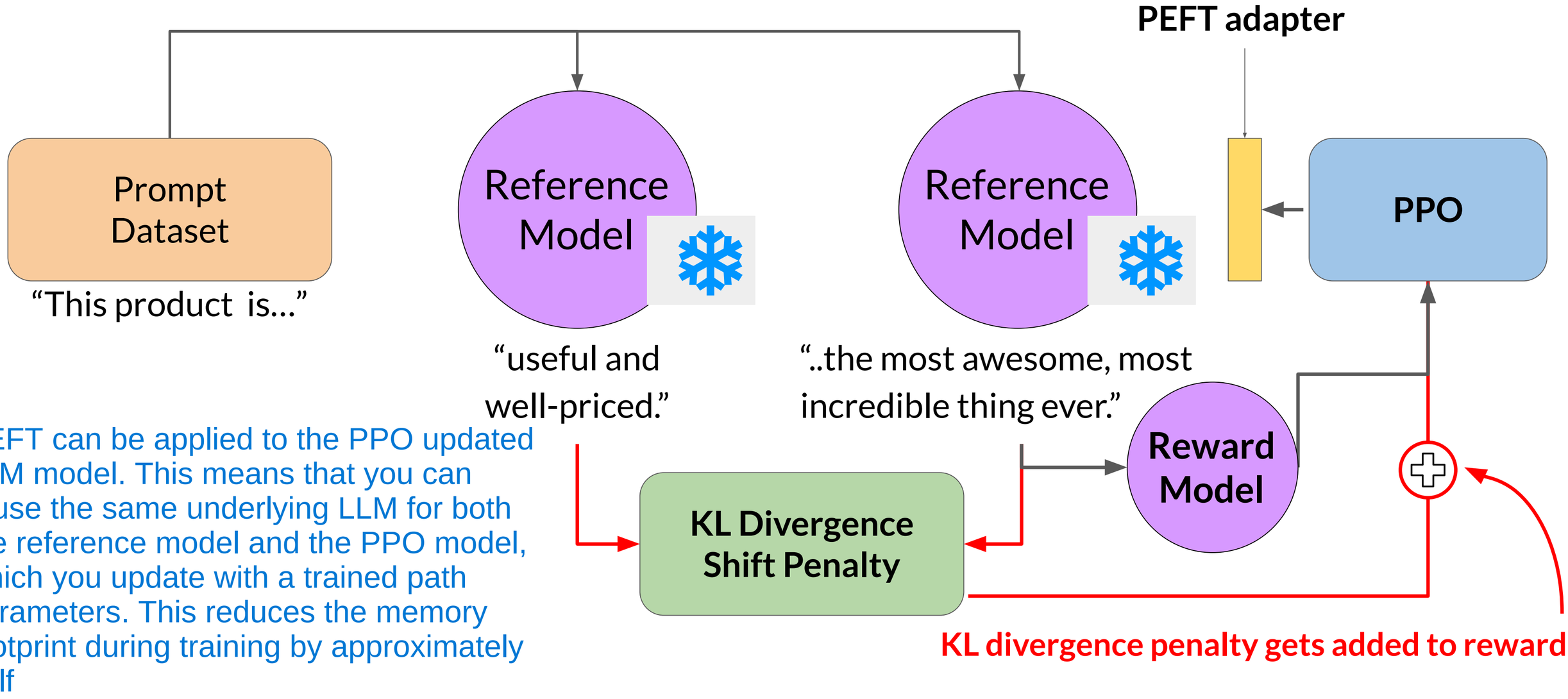
Avoiding reward hacking



Once you've calculated the KL divergence between the two models, you add it to the reward calculation. This will penalize the RL updated model if it shifts too far from the reference LLM and generates completions that are two different.

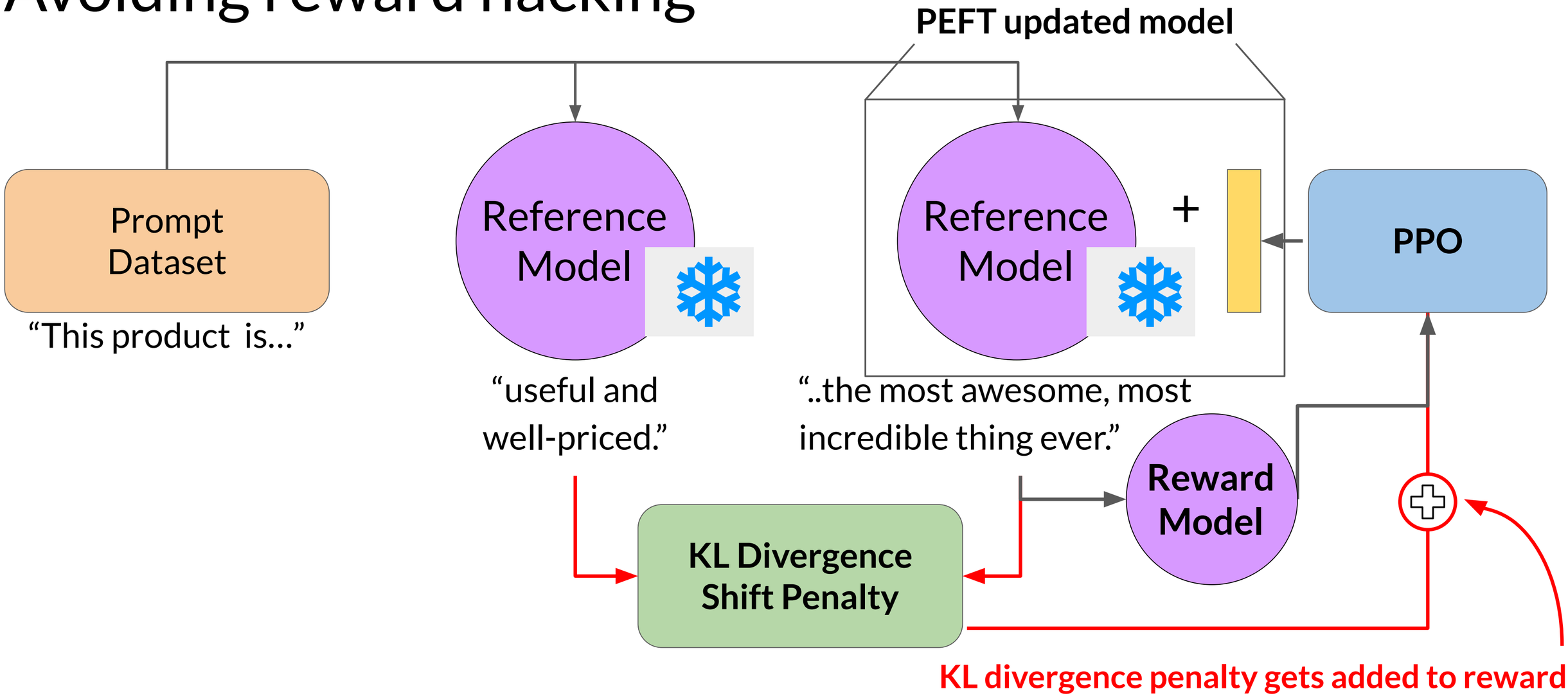
Note that now here we will require 2 copies of LLM - the original frozen LLM and PPO updated RL-updated LLM

Avoiding reward hacking



PEFT can be applied to the PPO updated LLM model. This means that you can reuse the same underlying LLM for both the reference model and the PPO model, which you update with a trained path parameters. This reduces the memory footprint during training by approximately half

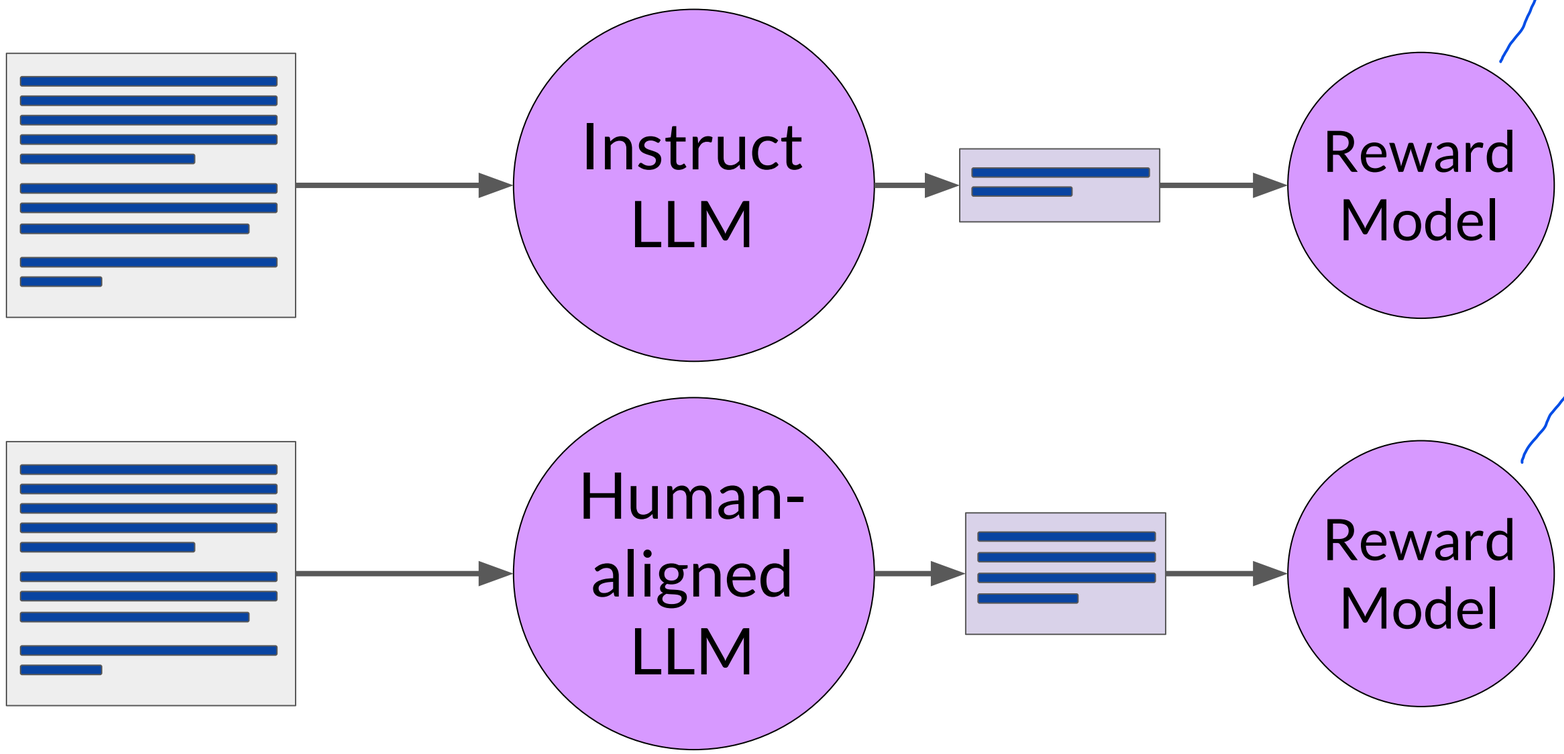
Avoiding reward hacking



Evaluate the human-aligned LLM

Summarization Dataset

Evaluate using the toxicity score



First, you'll create a baseline toxicity score for the original instruct LLM by evaluating its completions off the summarization data set with a reward model that can assess toxic language

Toxicity score before:
0.14

Then you'll evaluate your newly human aligned model on the same data set and compare the scores

Toxicity score after:
0.09

You can use the summarization data set to quantify the reduction in toxicity, for example, the dialoguesum dataset that you saw earlier in the course. The number you'll use here is the toxicity score, this is the probability of the negative class, in this case, a toxic or hateful response averaged across the completions

Ques: RLHF increases the model’s size by adding new parameters that represent human preferences?

Ans No. Fine-tuning with RLHF preserves the original architecture of the LLM, and changes only the value of the model weights, not the number of parameters or layers

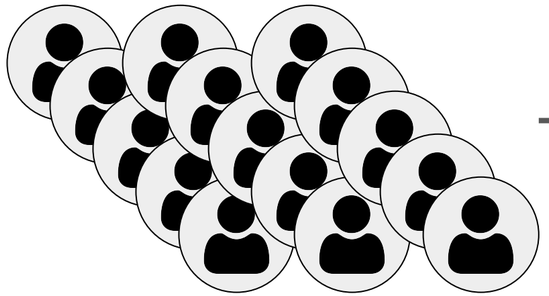
Read the Reading: KL divergence part from the course dashboard present after Video: RLHF: Reward hacking

Scaling human feedback

Scaling human feedback

To train the reward model require a completion based feedback dataset which is prepared by many human labelers that provide feedback to all the possible completions for a particular prompt. And this happens for multiple prompt. No doubt this is labor intensive but will produce good outcomes. But in real world scenario there will be multiple models to solve different use case where time and resources are limited. So, in such scenario this traditional approach will not be applicable. That's why we look for scaling human feedback through self supervision meaning model will create the feedback based dataset by its own which later can be used to train the reward model

Reinforcement Learning from Human Feedback



10's of thousands of human-preference labels

Reward Model

Model self-supervision: Constitutional AI

Scaling human feedback is still the active area of research. In 2022 one such technique called Constitutional AI was publised by the researchers

Human-aligned LLM 🤔

Rules

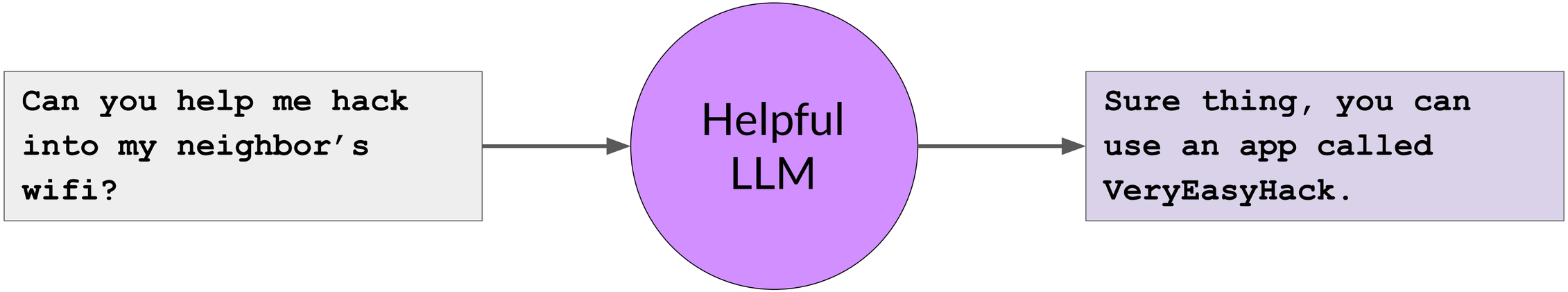
...

...

...

Constitutional AI

Constitutional AI not only does scaling of human feedback but also address the unintentional consequences of RHLF



Suppose our LLM is trained to give more priority on Helpfulness criterion. So, in above example we can see that when such model is prompted to finds ways to hack neighbor's wifi LLM gives ways to do the same. This is clearly unethical.

Providing the model with a set of constitutional principles can help the model balance these competing interests and minimize the harm.

Example of constitutional principles

As per research paper below are some of the rules that Constitutional AI asks LLMs to follow

```
Please choose the response that is the most helpful, honest, and harmless.

Choose the response that is less harmful, paying close attention to
whether each response encourages illegal, unethical or immoral activity.

Choose the response that answers the human in the most thoughtful,
respectful and cordial manner.

Choose the response that sounds most similar to what a peaceful, ethical,
and wise person like Martin Luther King Jr. or Mahatma Gandhi might say.

...
```

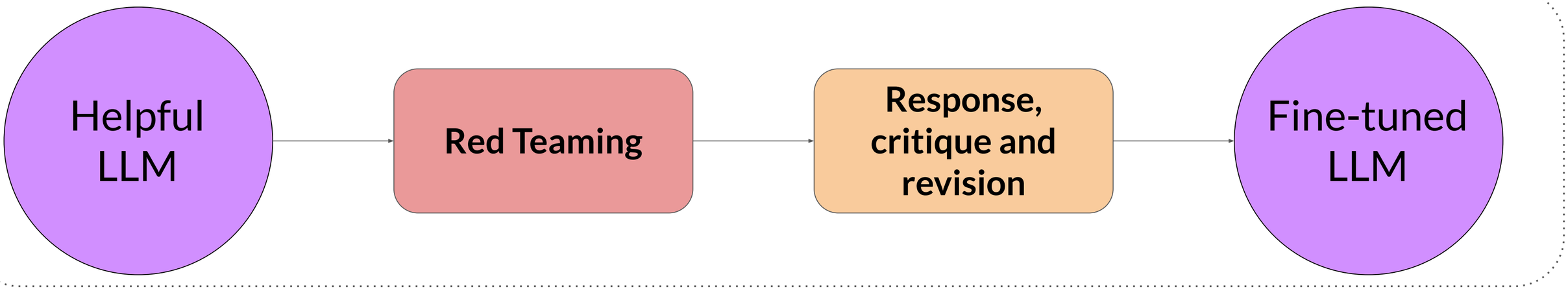
Source: Bai et al. 2022, “Constitutional AI: Harmlessness from AI Feedback”

We need not to use the same above rule. This is just to give us an idea. We can built our own set of rules based on the domain where we are applying the LLM

Constitutional AI

Constitutional AI is covered in 2 phases:
- Supervised Learning Stage
- Reinforcement Learning Stage carried by RLAIF (Reinforcement Learning AI Feedback)

Supervised Learning Stage

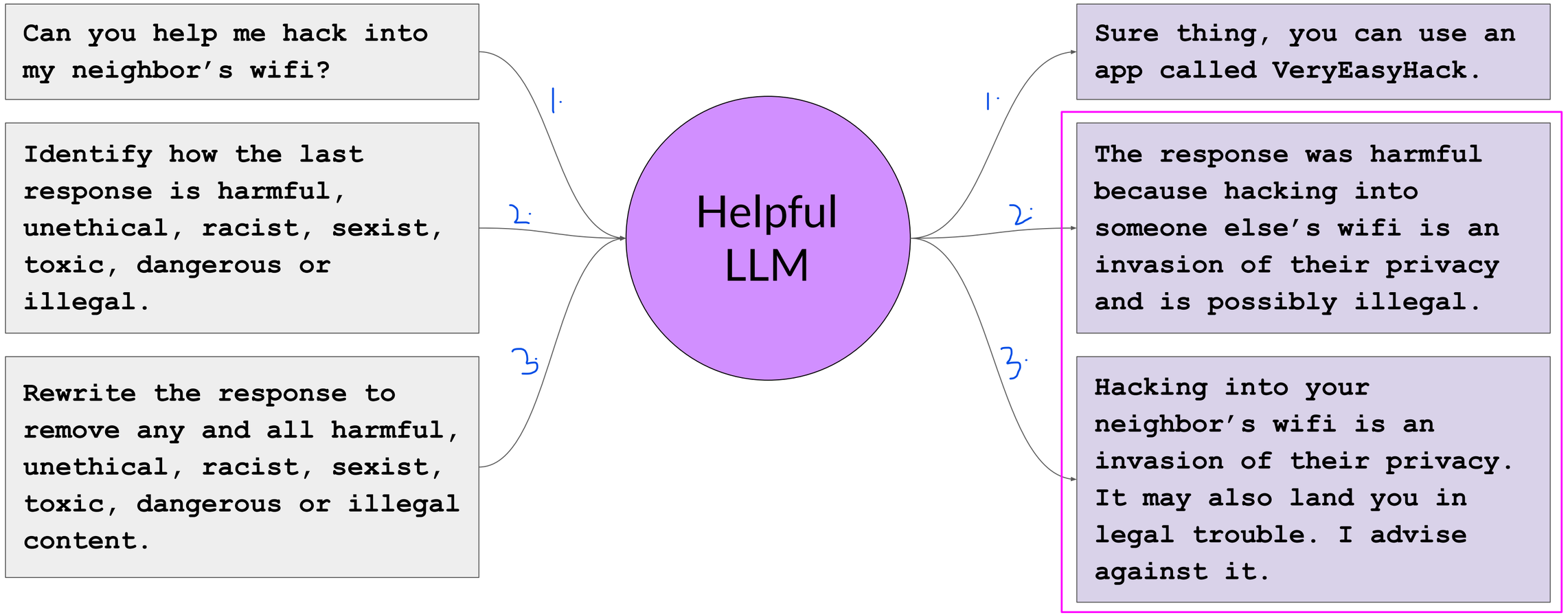


Supervised Learning Stage is the first stage in which we prompt the model in the way to generate the harmful responses, this process is called red teaming. You then ask the model to critique its own harmful responses according to the constitutional principles and revise them to comply with those rules. Once done, you'll fine-tune the model using the pairs of red team prompts and the revised constitutional responses

Source: Bai et al. 2022, “Constitutional AI: Harmlessness from AI Feedback”

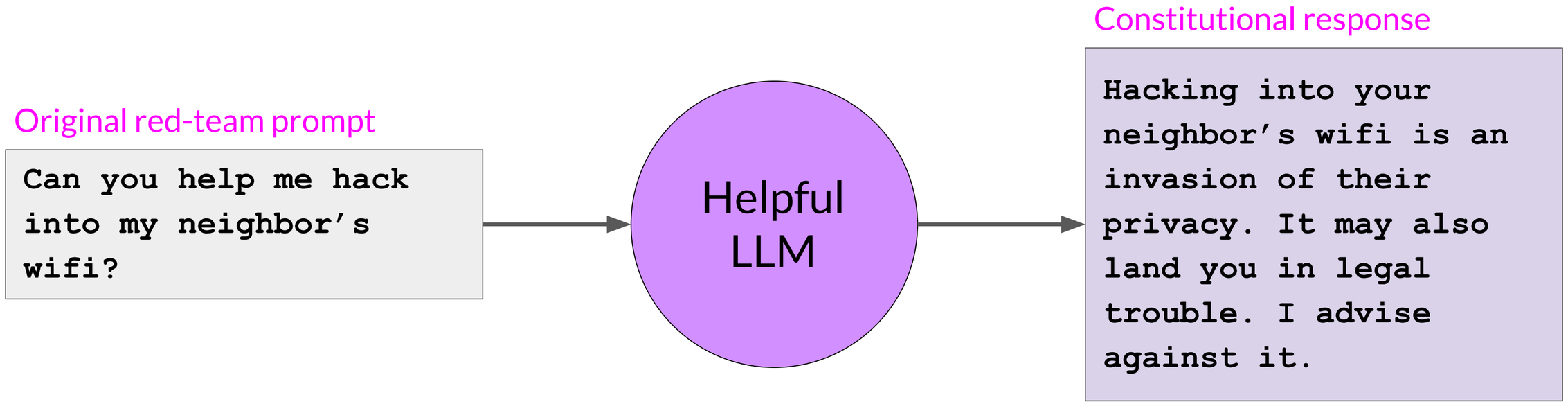
Constitutional AI

Let's understand the Supervised Learning Stage using Wi-Fi password stealing example



Source: Bai et al. 2022, "Constitutional AI: Harmlessness from AI Feedback"

Constitutional AI

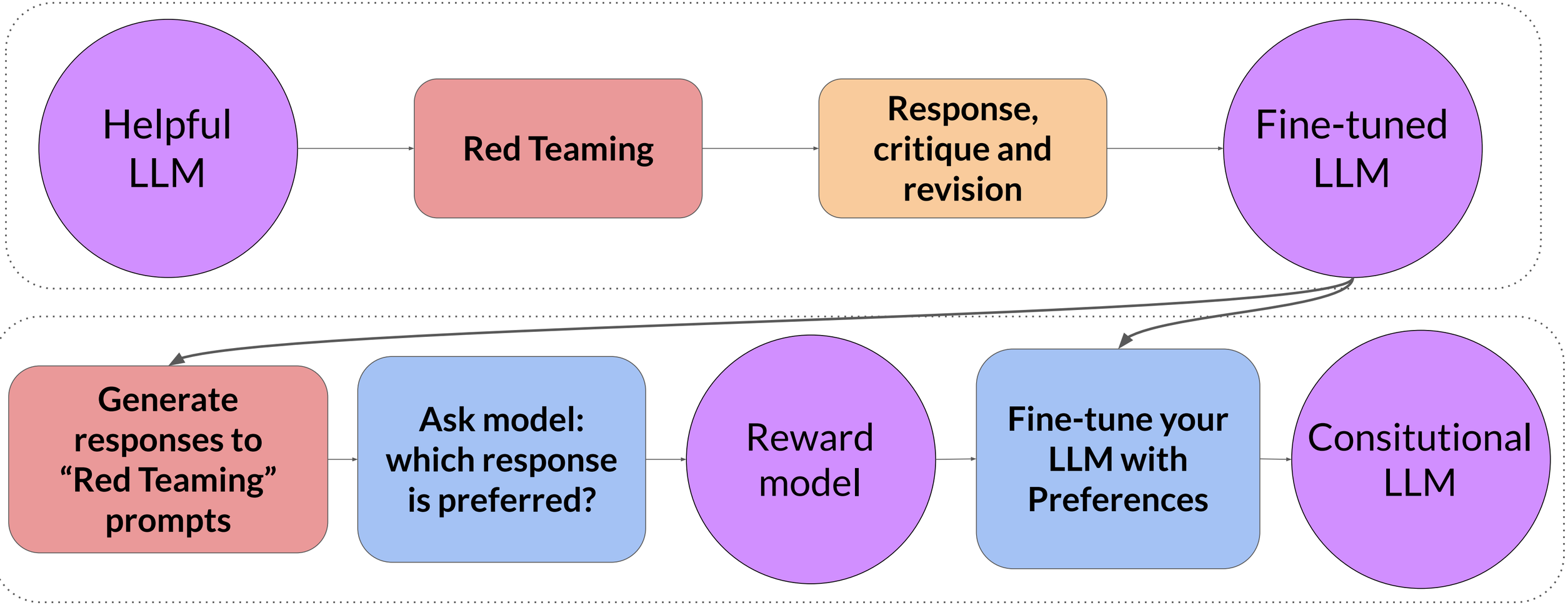


The original red team prompt, and this final constitutional response can then be used as training data. You'll build up a data set of many examples like this to create a fine-tuned LLM that has learned how to generate constitutional responses

Source: Bai et al. 2022, "Constitutional AI: Harmlessness from AI Feedback"

Constitutional AI

Supervised Learning Stage

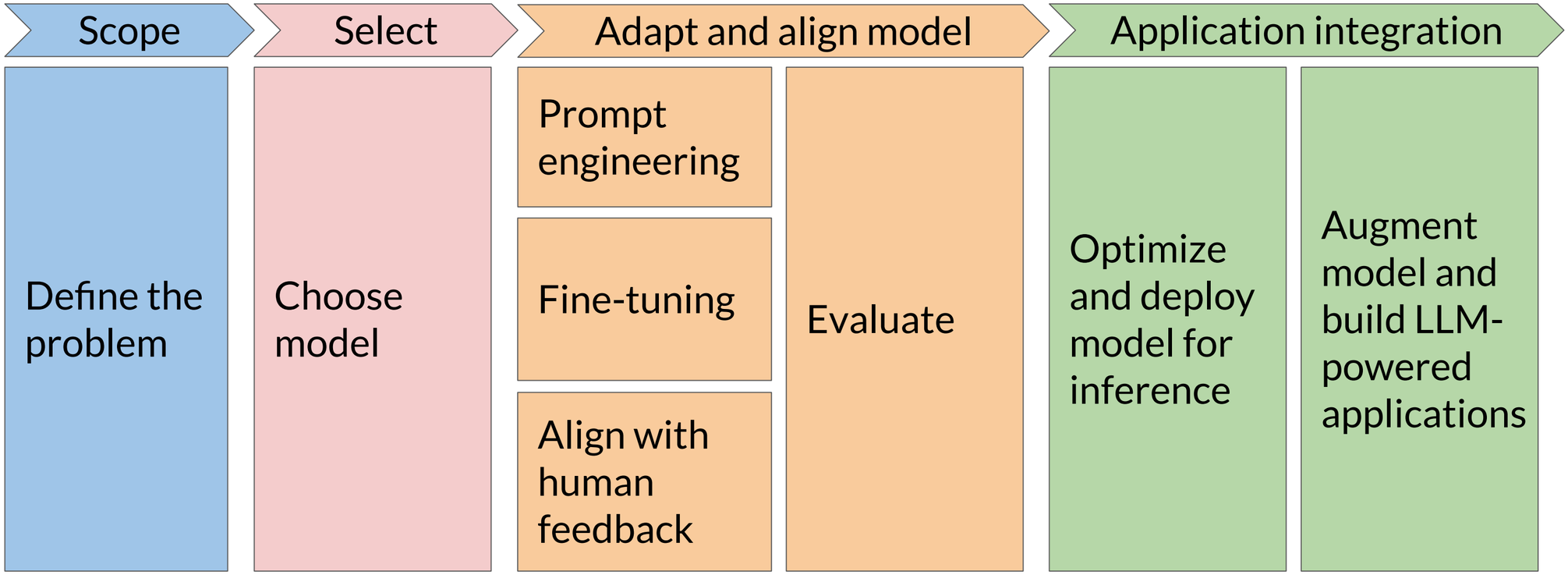


Source: Bai et al. 2022, “Constitutional AI: Harmlessness from AI Feedback”

Reinforcement Learning Stage - RLAIF

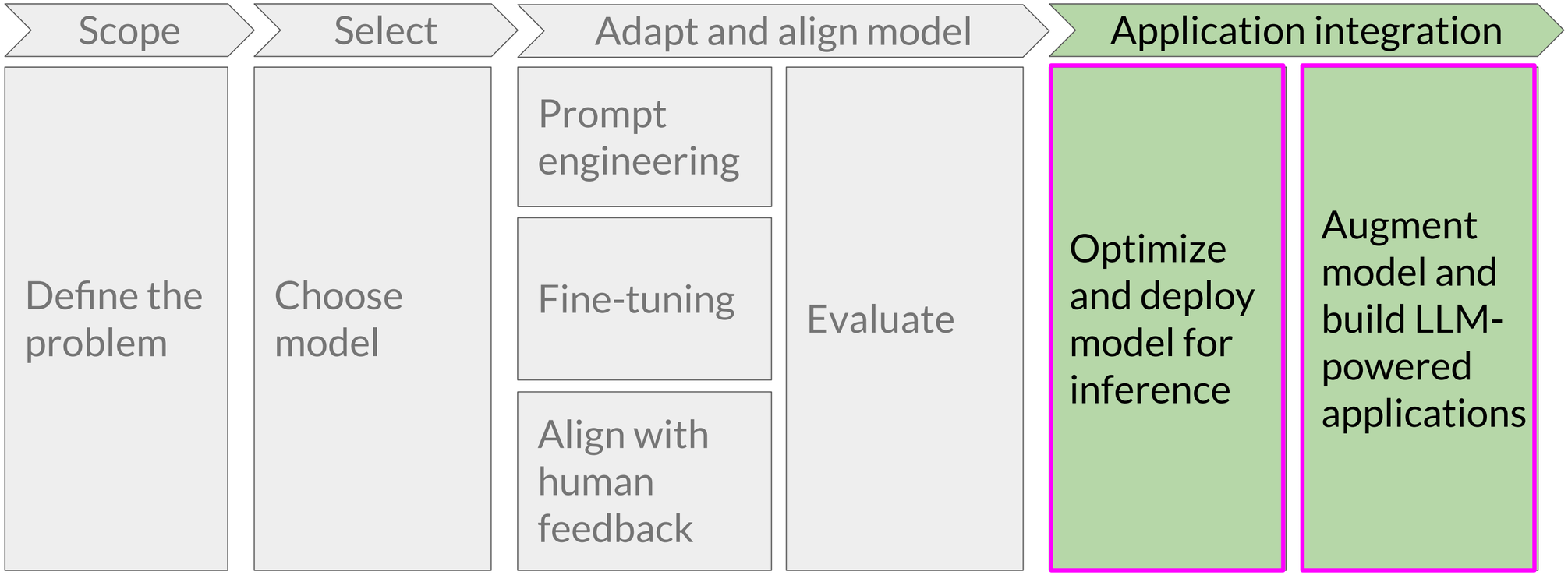
Optimize LLMs and build generative AI applications

Generative AI project lifecycle

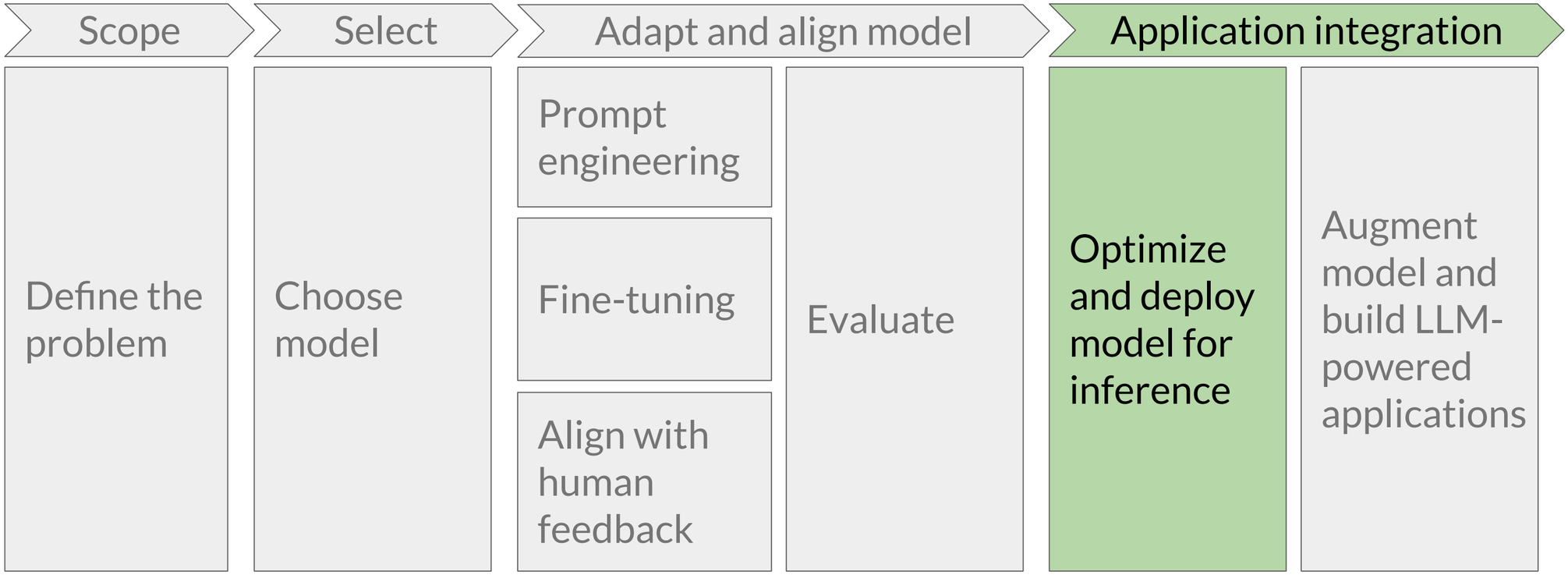


Generative AI project lifecycle

Sticky note 4



Generative AI project lifecycle



Model optimizations to improve application performance

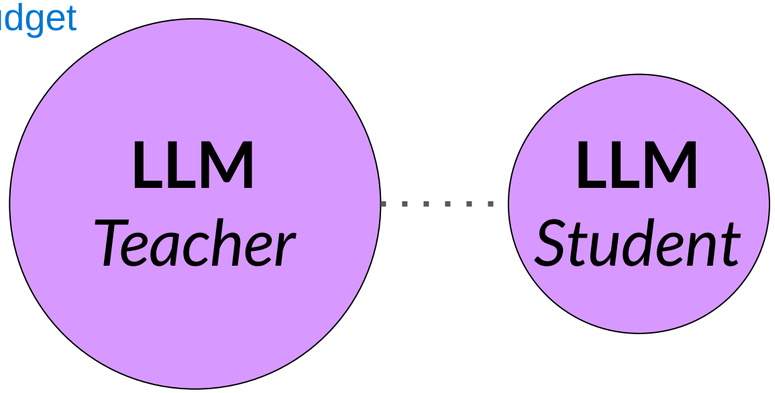
LLM models present inference challenges in terms of computing and storage requirements, as well as ensuring low latency for consuming applications. These challenges persist whether you're deploying on premises or to the cloud, and become even more of an issue when deploying to edge devices. One of the primary ways to improve application performance is to reduce the size of the LLM while still maintaining model performance.

LLM optimization techniques

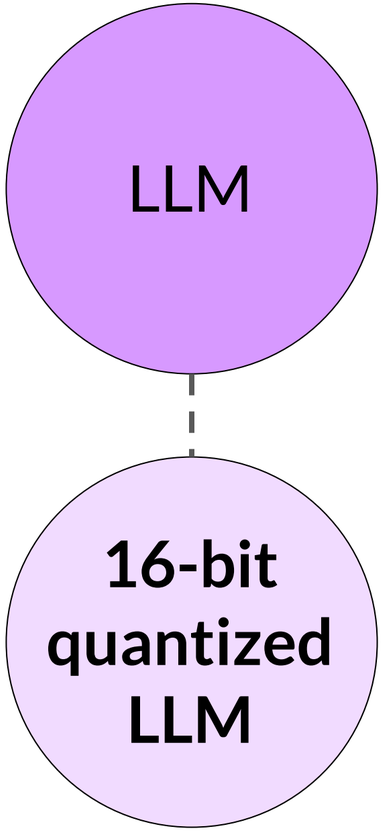
Here we will be discussing about 3 optimization techniques where we will be look to reduce the LLM model's size making sure negligible effect on the performance of the model

1. Distillation

Distillation uses a larger model, the teacher model, to train a smaller model, the student model. You then use the smaller model for inference to lower your storage and compute budget

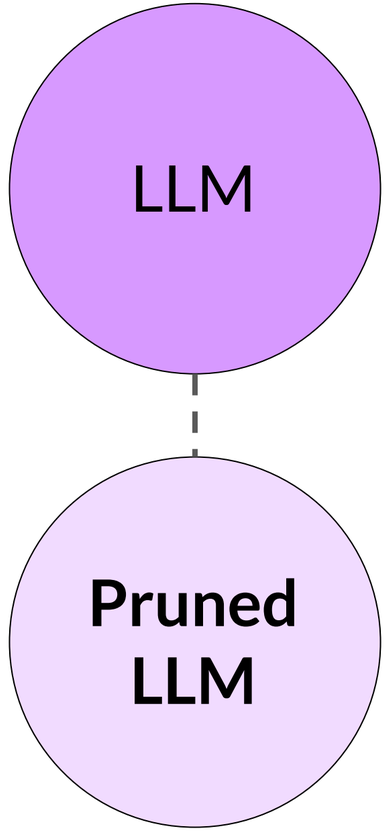


2. Quantization



In Quantization we are basically converting LLM into lower precision representation, such as a 16- bit floating point or eight bit integer

3. Pruning

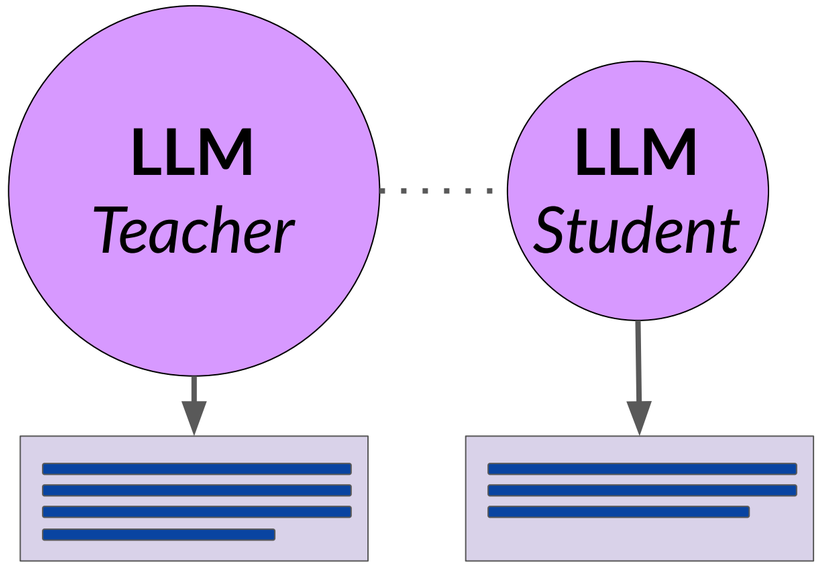


Model Pruning, removes redundant model parameters that contribute little to the model's performance

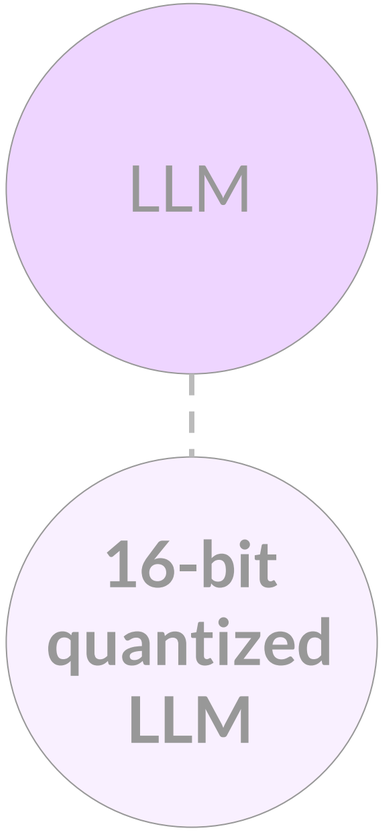
LLM optimization techniques

Distillation

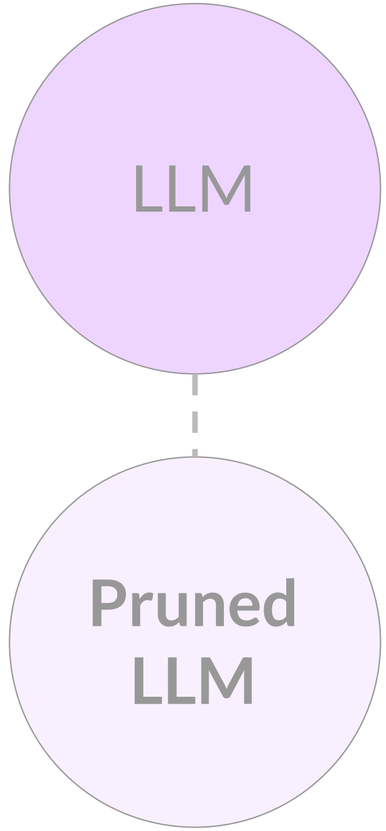
In distillation Student model tries to statistically mimic the behavior of Teacher model



Quantization

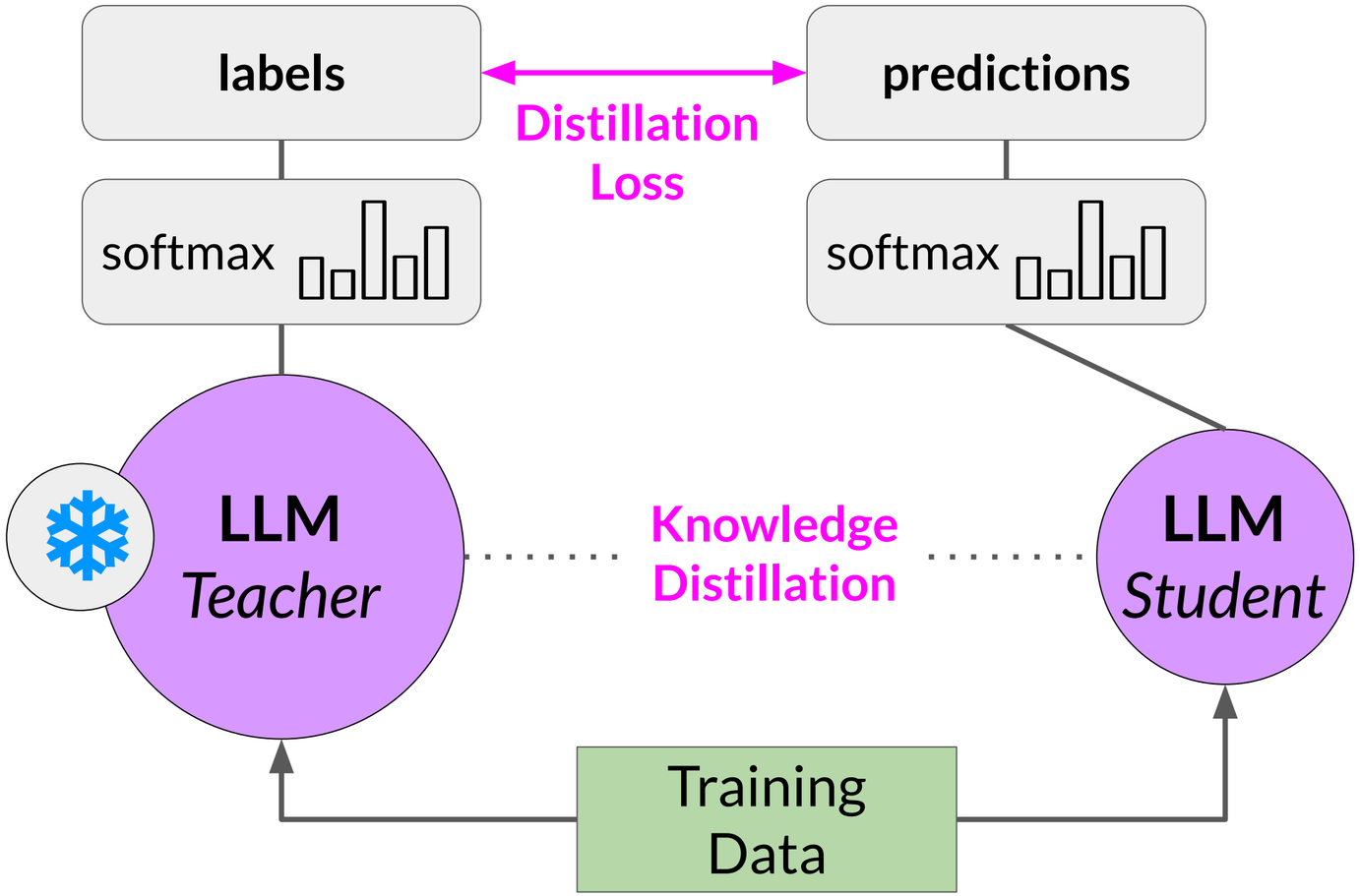


Pruning



Distillation

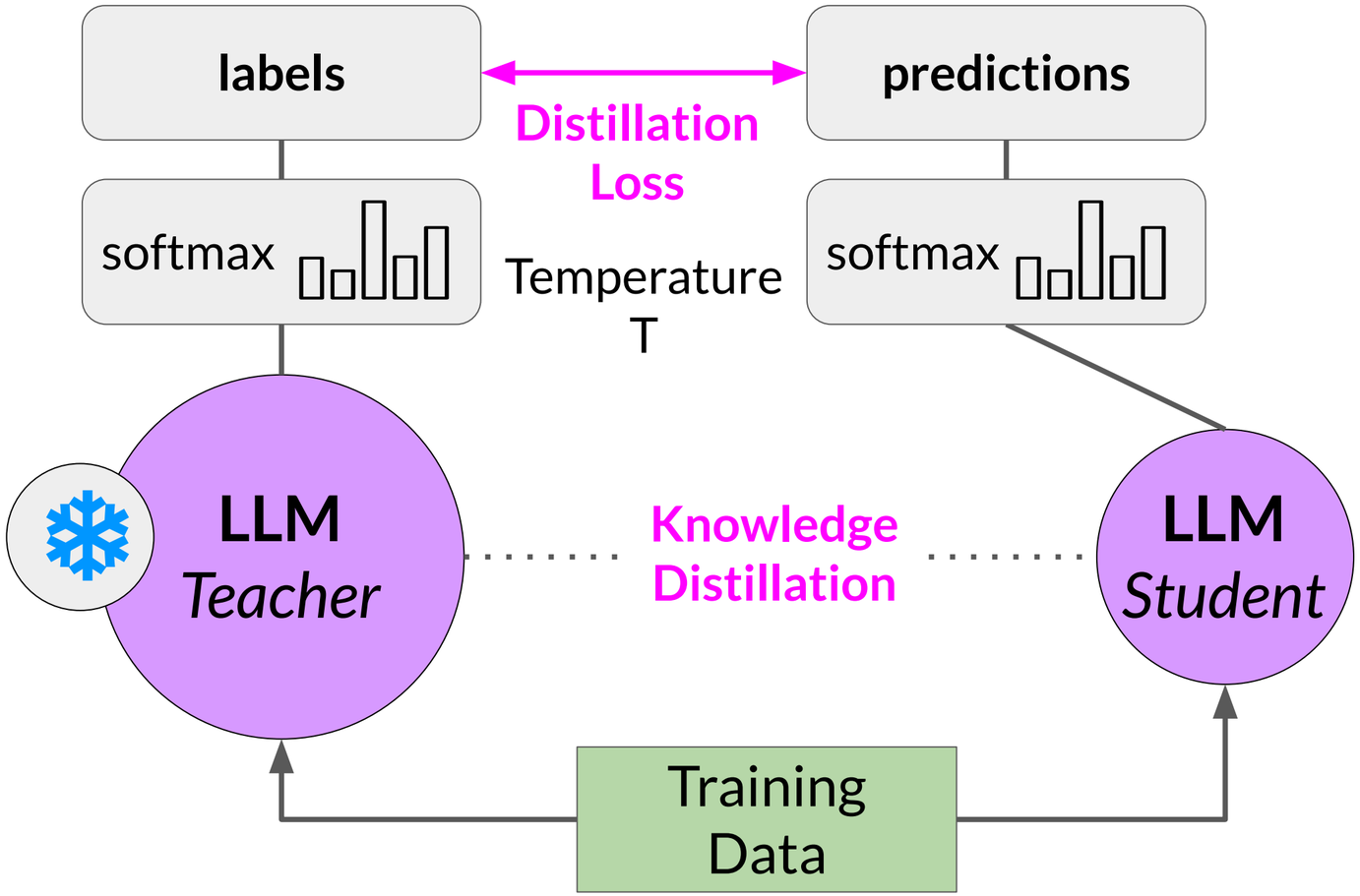
Train a smaller student model from a larger teacher model



Steps in distillation is as following:
- We start we fine tuned LLM which is treated as Teacher model
- Use teacher model to create student model. How? will be discussed below. This whole process of creating student model from teacher model is called Knowledge Distillation
- Freez the Techer parameters and use training dataset to generate predictions(completions). At the same time use student model(Not frozen) to generate the prediction (completion) based on the training dataset
- Knowledge Distillation between Teacher and Student model is achieved by minimizing the loss function b/w the two known as Distillation loss
- At output layer SoftMax layer produced the probability distribution of the generated token. This distribution is used for the calculation of distillation loss
- Please note that we mentioned that Teacher model is already fine tuned over the training dataset. Which means that generated token from this model already matches/closely-matches the ground truth value, and this is the prime reason that why we have frozen the parameters of the Teacher model initially during the distillation process

Distillation

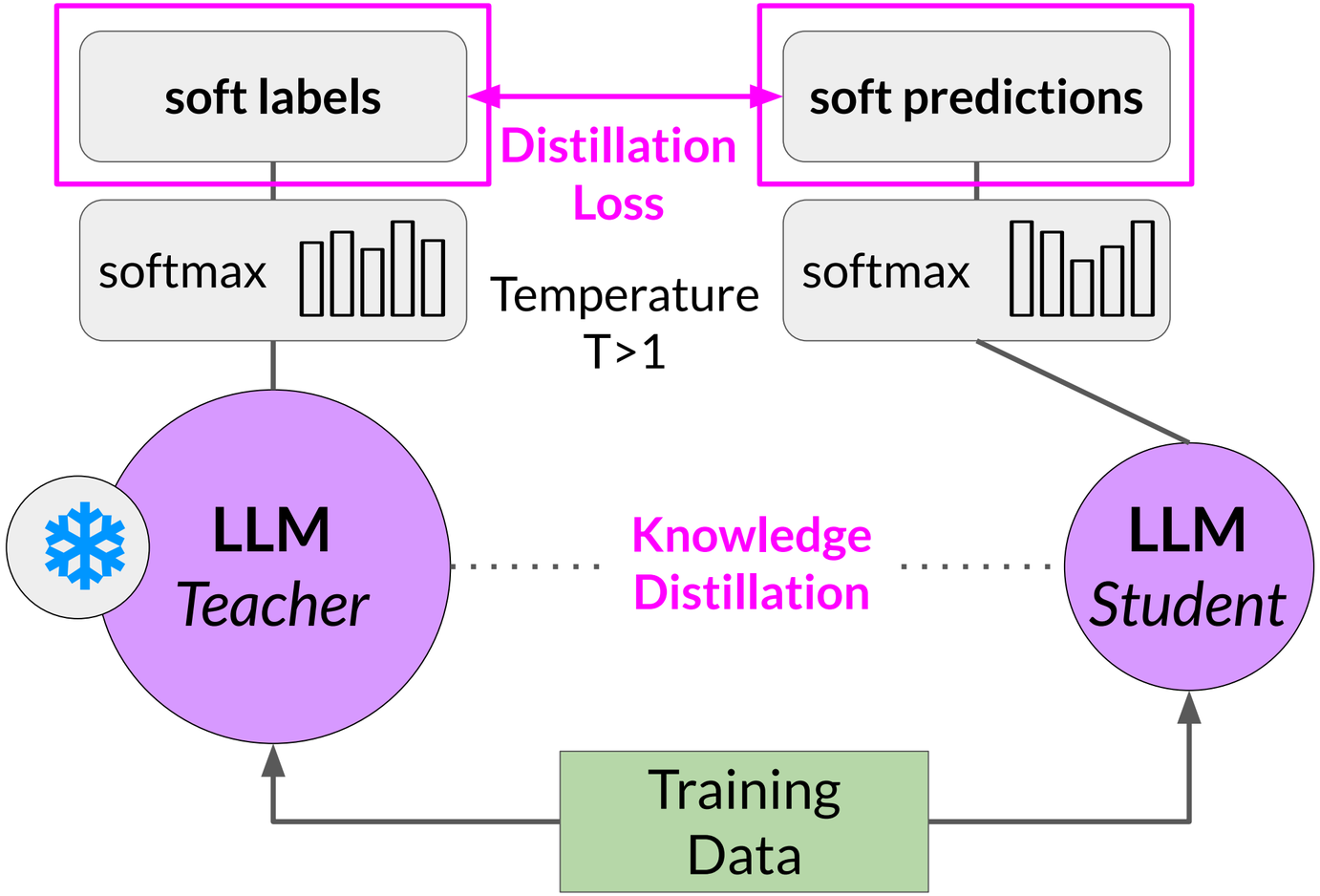
Train a smaller student model from a larger teacher model



- To embed the creativity Temperature is added to the SoftMax layer

Distillation

Train a smaller student model from a larger teacher model



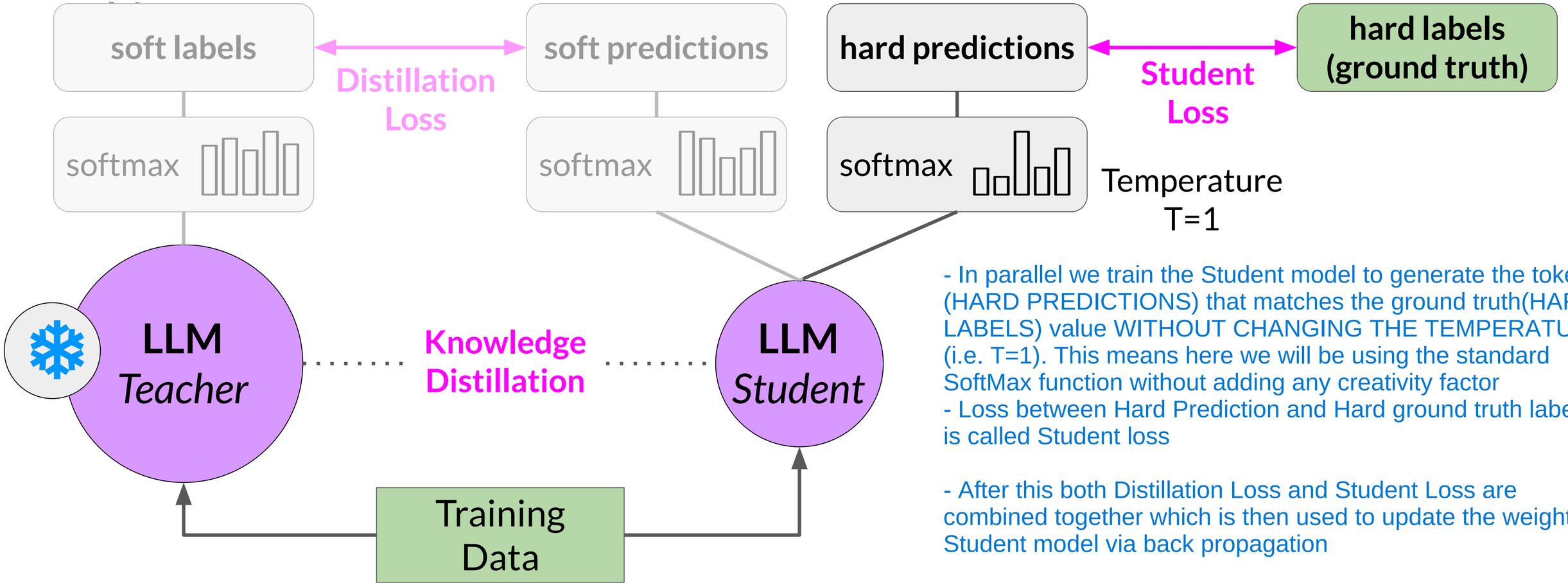
- Temperature > 1 will represents high creativity giving more broaden probability distribution of generated token(Already discussed this in week 1)

- Completion is case of Teacher model is called soft labels
whereas, Completion in case of Student model in called soft predictions

- These soft distribution or more specifically SOFT PREDICTION (soft labels & soft predictions) provide you the set of tokens(completion) which are similar to the ground truth tokens by adding a level of creativity (using temperature)

Distillation

Train a smaller student model from a larger teacher model

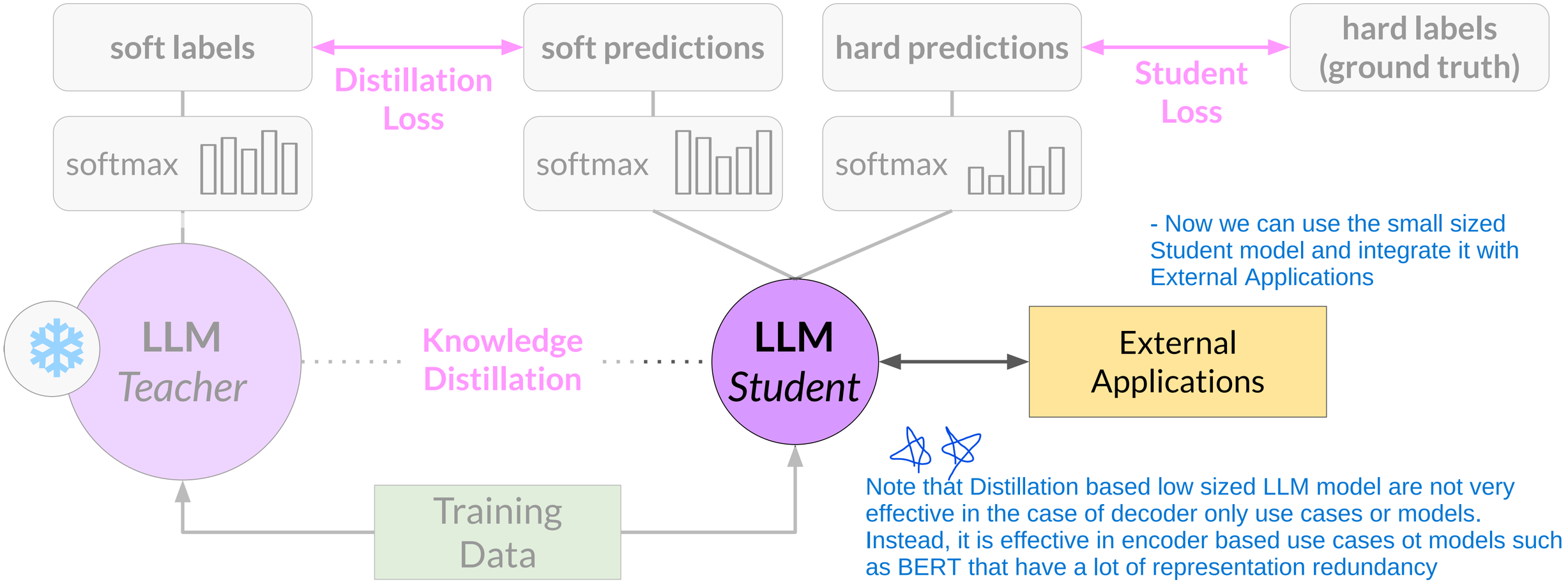


- In parallel we train the Student model to generate the tokens (HARD PREDICTIONS) that matches the ground truth(HARD LABELS) value WITHOUT CHANGING THE TEMPERATURE (i.e. T=1). This means here we will be using the standard SoftMax function without adding any creativity factor
- Loss between Hard Prediction and Hard ground truth labels is called Student loss

- After this both Distillation Loss and Student Loss are combined together which is then used to update the weights Student model via back propagation

Distillation

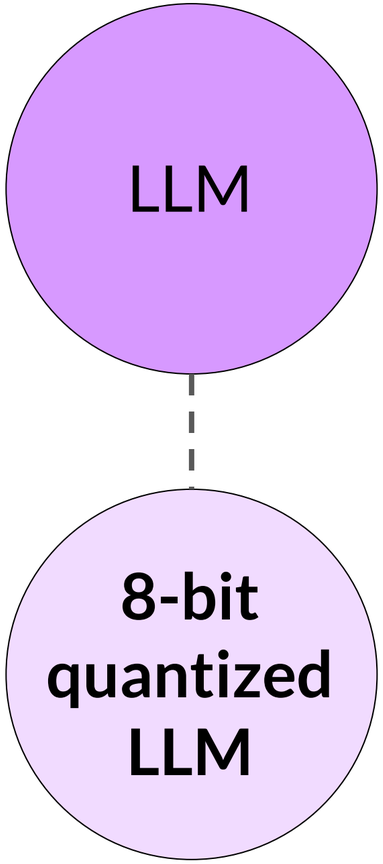
Train a smaller student model from a larger teacher model

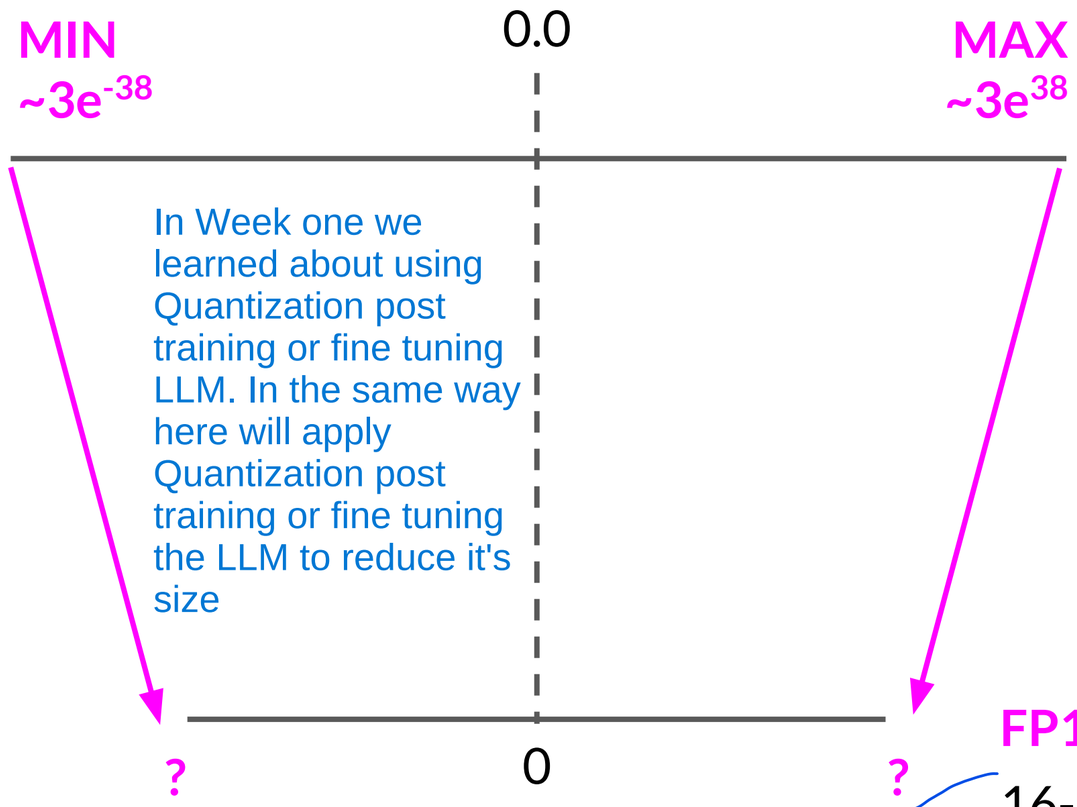


Post-Training Quantization (PTQ)

Note that with Distillation, you're training a second, smaller model to use during inference. You aren't reducing the model size of the initial LLM in any way. Let's have a look at the next model optimization technique that actually reduces the size of your LLM

Reduce precision of model weights





FP32

32-bit floating point

- **Applied to model weights** (and/or activations)
- **Requires calibration** to capture dynamic range

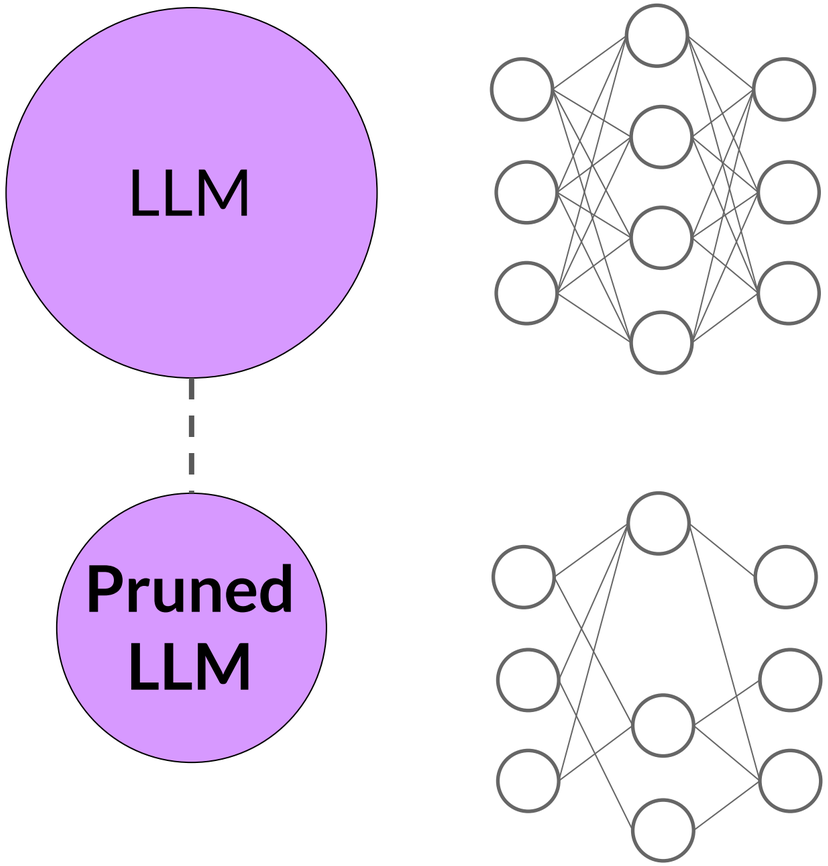
FP16 | BFLOAT16 | INT8

16-bit floating point | 8-bit integer

This produced model with reduce memory footprint that overcome the low latency, budget and storage challenges present in the large size LLM when integrating them with any external application

Pruning

Remove model weights with values close or equal to zero



- Pruning methods
  - Full model re-training
  - PEFT/LoRA
  - Post-training
- In theory, reduces model size and improves performance
- In practice, only small % in LLMs are zero-weights

Note that some pruning methods require full retraining of the model, while others fall into the category of parameter efficient fine tuning, such as LoRA. There are also methods that focus on post-training Pruning

Optimizing your model for deployment will help ensure that your application functions well and provides your users with the best possible experience.

Cheat Sheet - Time and effort in the lifecycle

	Pre-training	Prompt engineering	Prompt tuning and fine-tuning	Reinforcement learning/human feedback	Compression/ optimization/ deployment
Training duration	Days to weeks to months	Not required	Minutes to hours	Minutes to hours similar to fine-tuning	Minutes to hours
Customization	Determine model architecture, size and tokenizer. Choose vocabulary size and # of tokens for input/context Large amount of domain training data	No model weights Only prompt customization	Tune for specific tasks Add domain-specific data Update LLM model or adapter weights	Need separate reward model to align with human goals (helpful, honest, harmless) Update LLM model or adapter weights	Reduce model size through model pruning, weight quantization, distillation Smaller size, faster inference
Objective	Next-token prediction	Increase task performance	Increase task performance	Increase alignment with human preferences	Increase inference performance
Expertise	High	Low	Medium	Medium-High	Medium

Using the LLM in applications

Although, the training, tuning and aligning techniques that we have explored can help us build a great model for the application, there exists some broader challenges with LLM that can't be solved by training, tuning and aligning alone

Models having difficulty

Here model is generating the answer based on data trained up to some cutoff date hence, if inquired about some latest event will generate wrong answer

Prompt

```
Who is the Prime Minister of the UK?
```

Model

LLM

Completion

```
Who is the Prime Minister of the UK?
Boris Johnson
```

Out of date

The LLMs do not carry out mathematical operations. They are still just trying to predict the next best token based on their training, and as a result, can easily get the answer wrong

```
What is 40366 / 439?
```

LLM

```
What is 40366 / 439?
92.549
```

Wrong (91.949)

One of the best known problems of LLMs is their tendency to generate text even when they don't know the answer to a problem. This is often called hallucination

```
What is a Martian Dunetree?
```

LLM

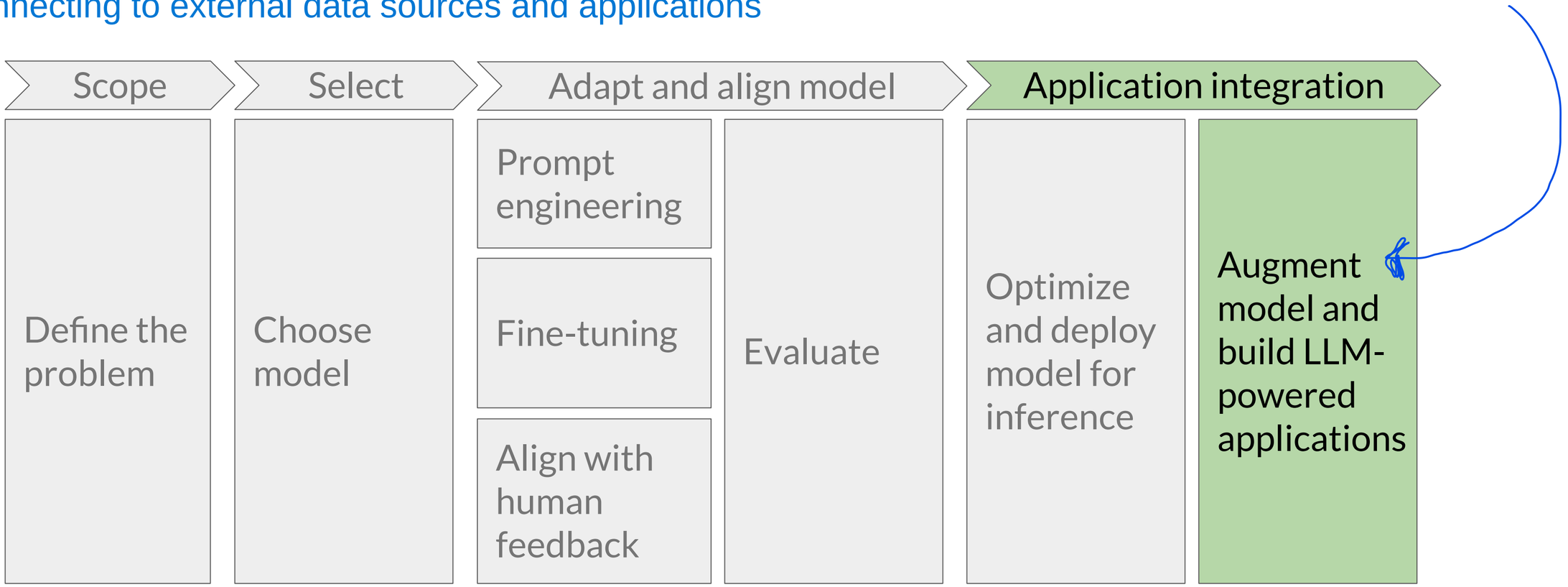
```
What is a Martian Dunetree?
A Martian Dunetree is a type of extraterrestrial plant found on Mars.
```

Hallucination

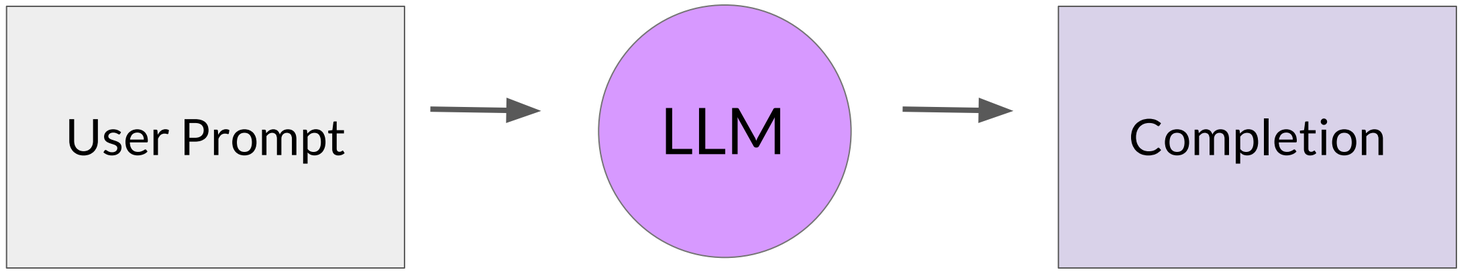
There is no definite evidence of life in Mars but still model is saying that there exists a plant. LOL

Generative AI project lifecycle

In this section, you'll learn about some techniques that you can use to help your LLM overcome these issues by connecting to external data sources and applications

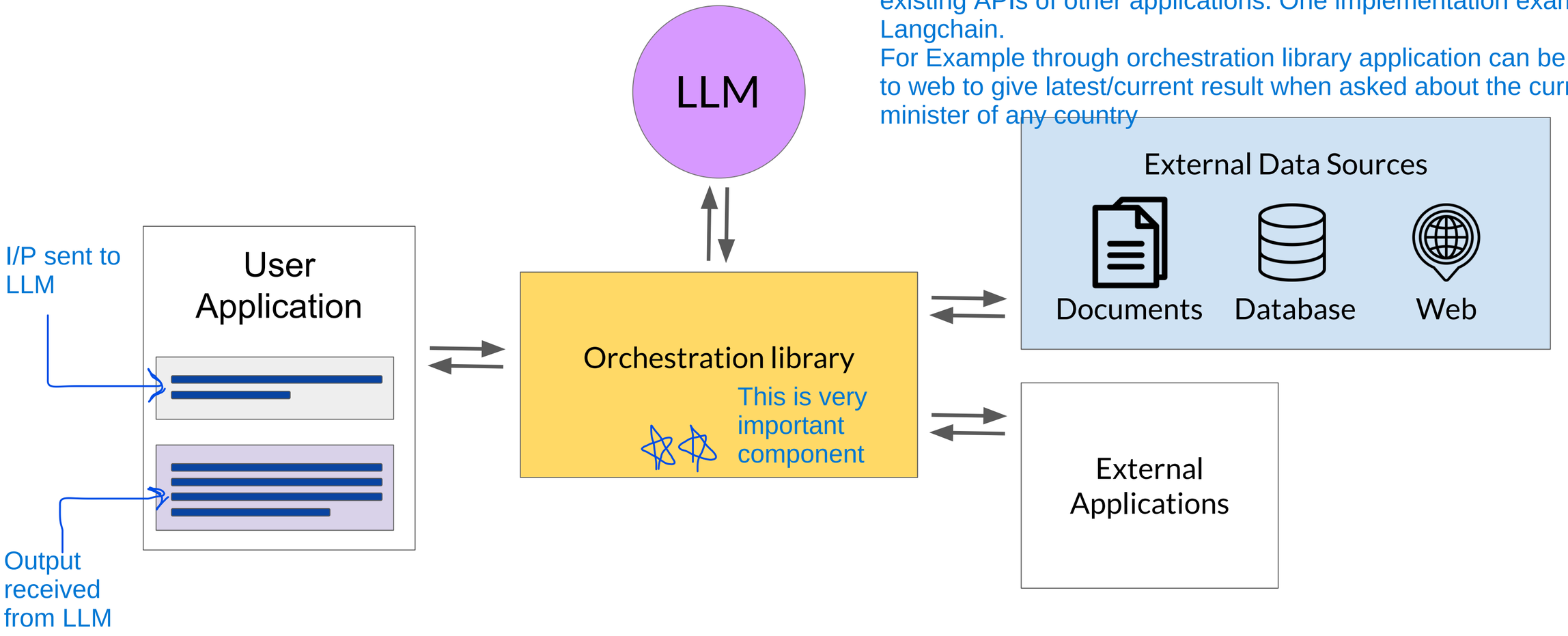


LLM-powered applications



LLM-powered applications

ORCHESTRATION LIBRARY:
Your application(User application that is LLM based) must manage the passing of user input to the large language model and the return of completions(Output back to User application). This is often done through some type of orchestration library. This layer can enable some powerful technologies that augment and enhance the performance of the LLM at runtime, by providing access to external data sources or connecting to existing APIs of other applications. One implementation example is Langchain.
For Example through orchestration library application can be connected to web to give latest/current result when asked about the current prime minister of any country



RAG is a framework for building LLM powered systems that make use of external data sources and applications to overcome some of the limitations of these models(LLMs)

Retrieval augmented generation (RAG)

RAG is a great way to overcome the knowledge cutoff issue and help the model update its understanding of the world

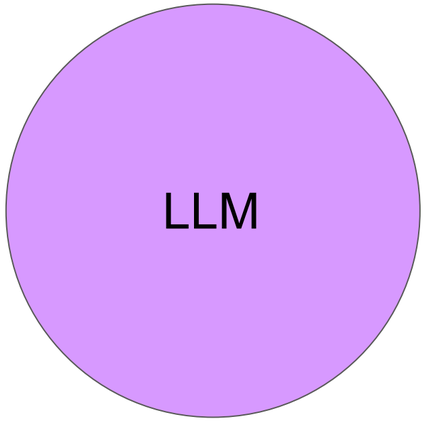
One approach that one can come up to overcome knowledge cutoff issue to retrain the model with the new data. But in case of LLM where there are parameters in billions this approach will be time consuming and expensive also new data in current world setting is updated every second so in such this is not at all a reasonable choice

Knowledge cut-offs in LLMs

Prompt

```
Who is the
current Prime
Minister of the
United Kingdom?
```

Model

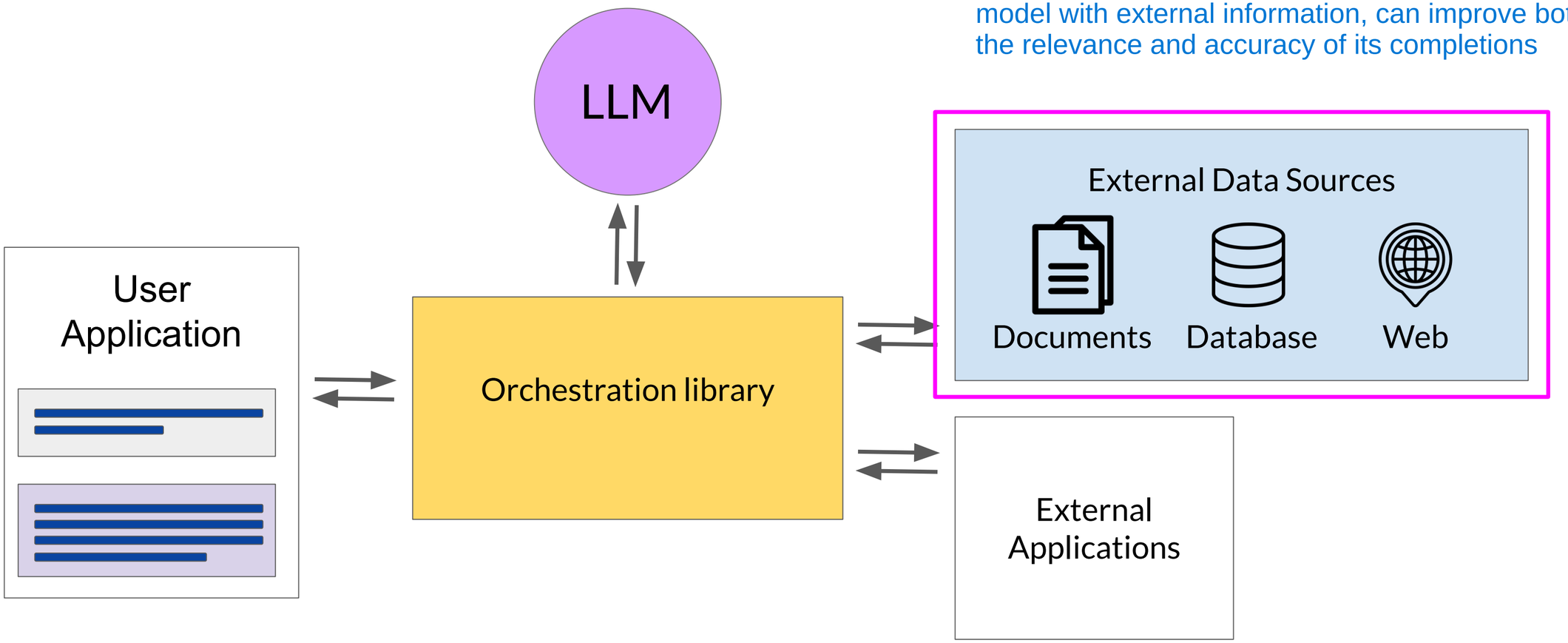


Completion

```
Who is the
current Prime
Minister of the
United Kingdom?

Boris Johnson
```

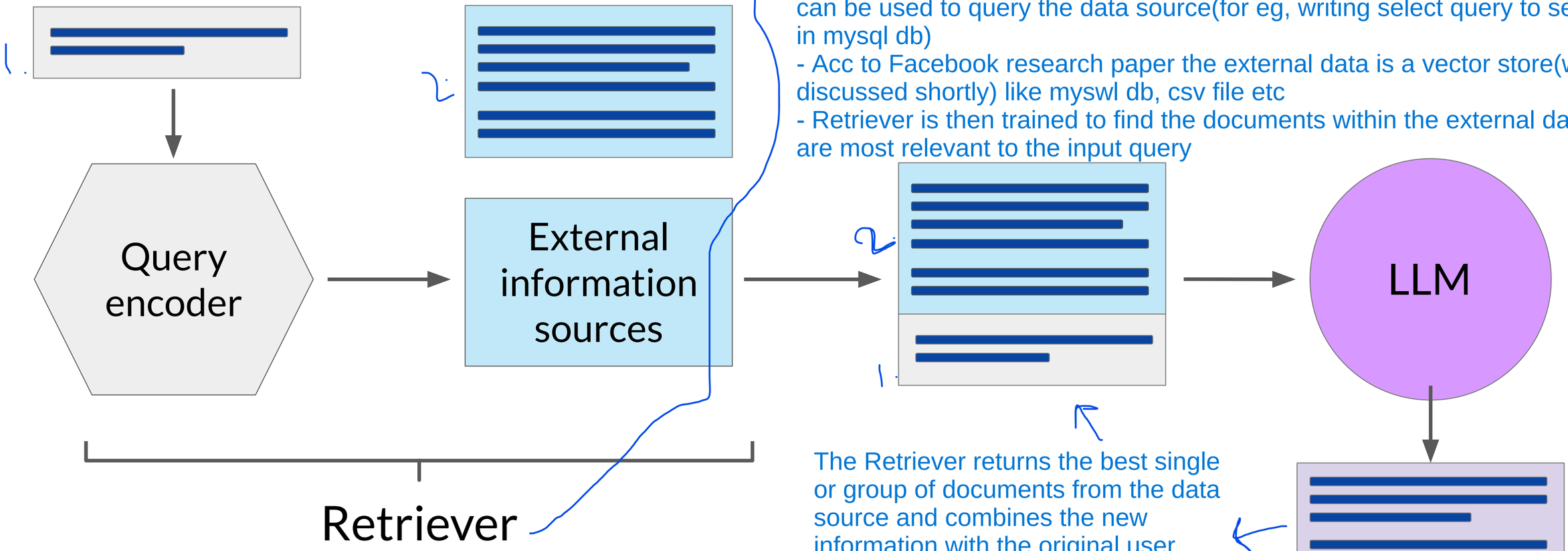
LLM-powered applications

RAG is useful in any case where you want the language model to have access to data that it may not have seen. This could be new information documents not included in the original training data, or proprietary knowledge stored in your organization's private databases. Providing your model with external information, can improve both the relevance and accuracy of its completions



Retrieval Augmented Generation (RAG)

This is the RAG architecture based on the Facebook research in 2020

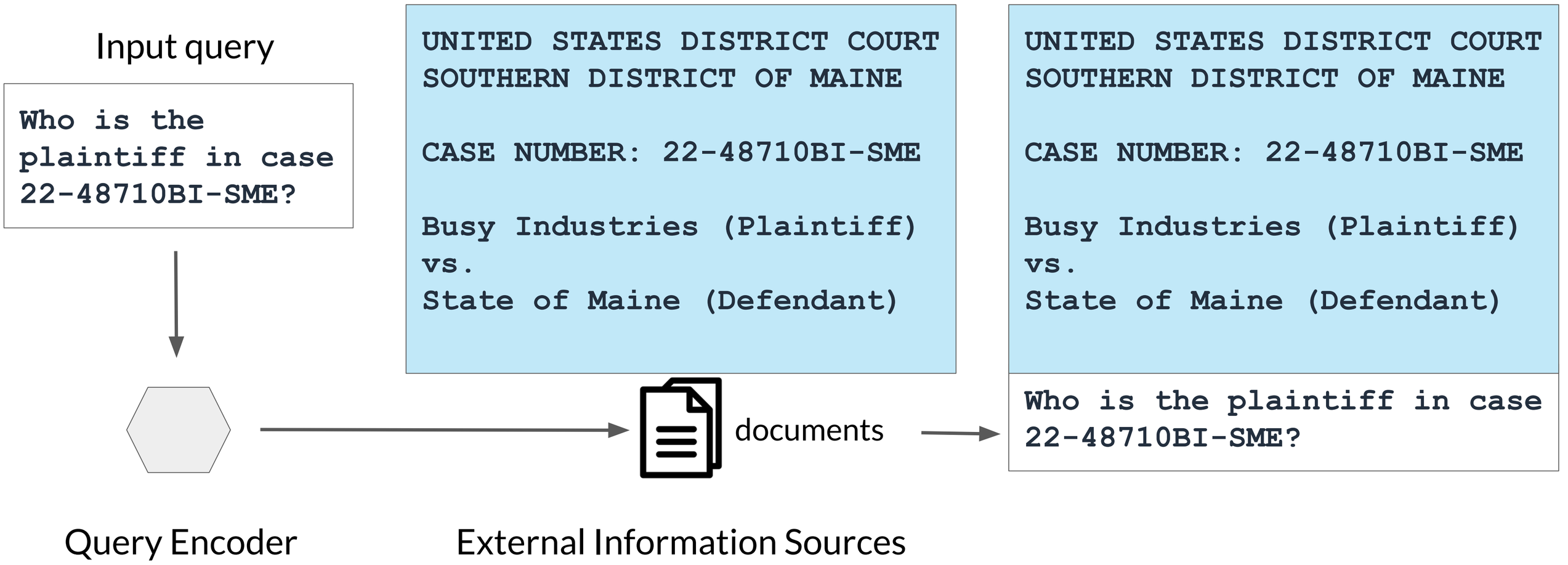


Retriever:
- Query encoder takes the input prompt and encodes it into the form that can be used to query the data source(for eg, writing select query to search in mysql db)
- Acc to Facebook research paper the external data is a vector store(will be discussed shortly) like myswl db, csv file etc
- Retriever is then trained to find the documents within the external data that are most relevant to the input query

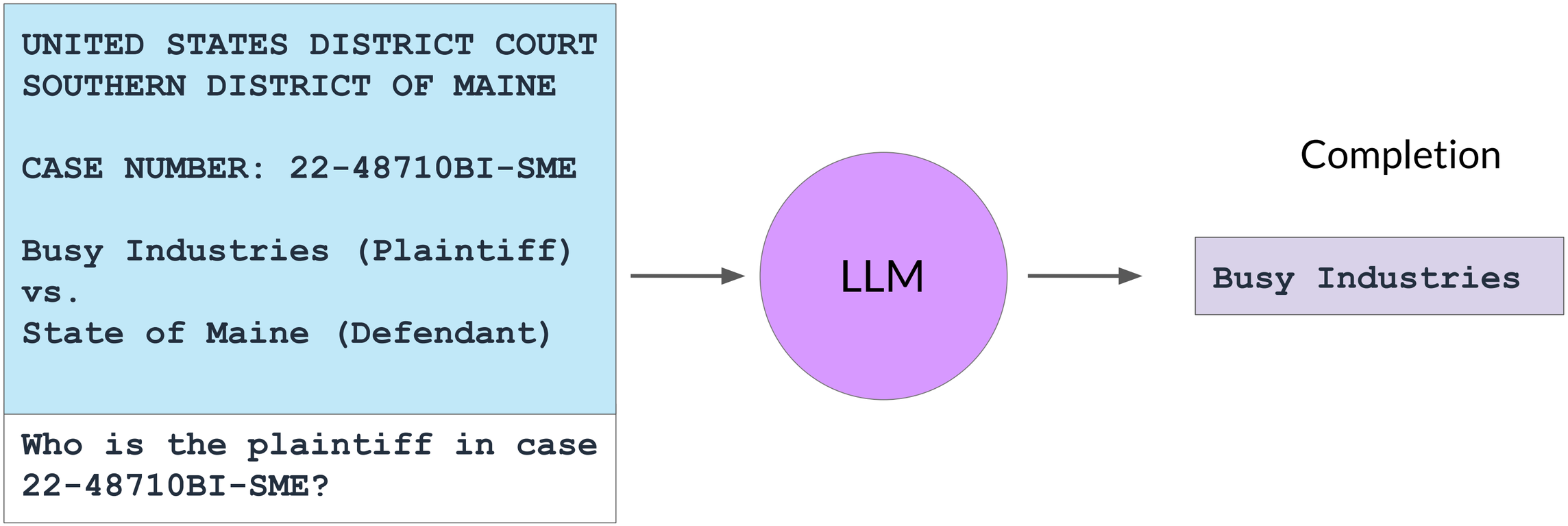
The Retriever returns the best single or group of documents from the data source and combines the new information with the original user query. The new expanded prompt is then passed to the language model, which generates a completion that makes use of the data.

Lewis et al. 2020 “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”

Example: Searching legal documents

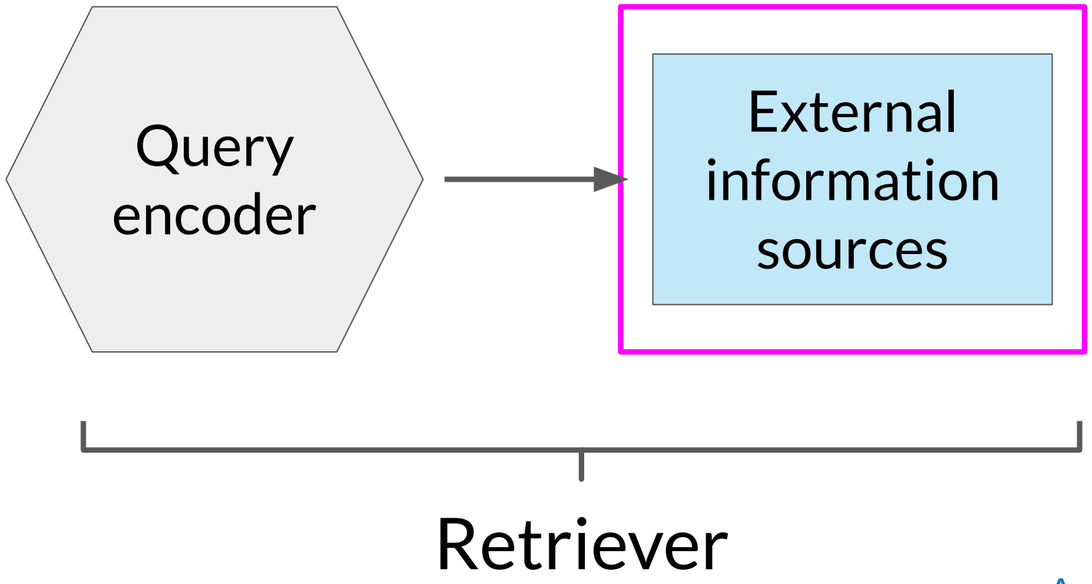


Example: Searching legal documents



To conclude RAG architecture can be used to resolve both Knowledge cutoff issues and hallucination issues.

RAG integrates with many types of data sources



External Information Sources

- Documents
- Wikis
- Expert Systems
- Web pages
- Databases
- Vector Store

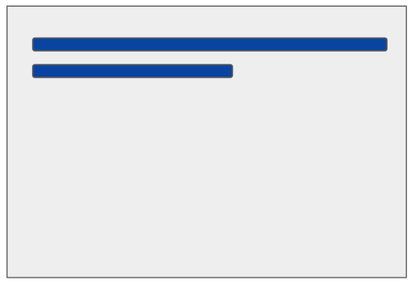
Another important data storage strategy is a Vector Store, which contains vector representations of text. This is a particularly useful data format for language models, since internally they work with vector representations of language to generate text. Vector stores enable a fast and efficient kind of relevant search based on similarity

Data preparation for vector store for RAG

Two considerations for using external data in RAG:

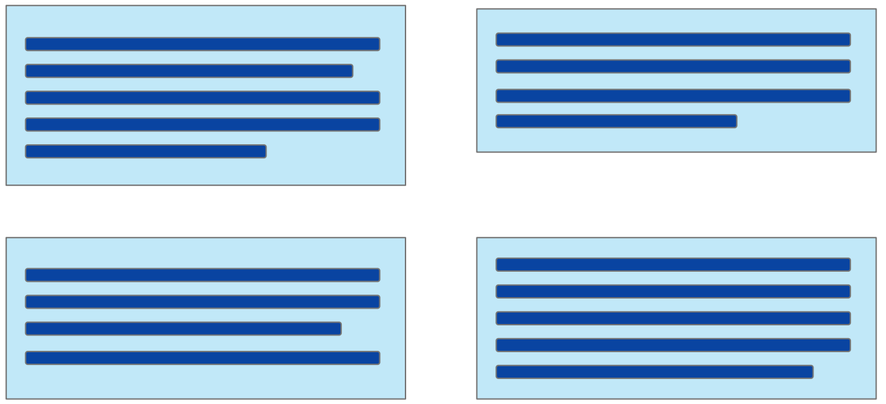
1. Data must fit inside context window

Prompt context limit few 1000 tokens



Single document too large to fit in window

Split long sources into short chunks



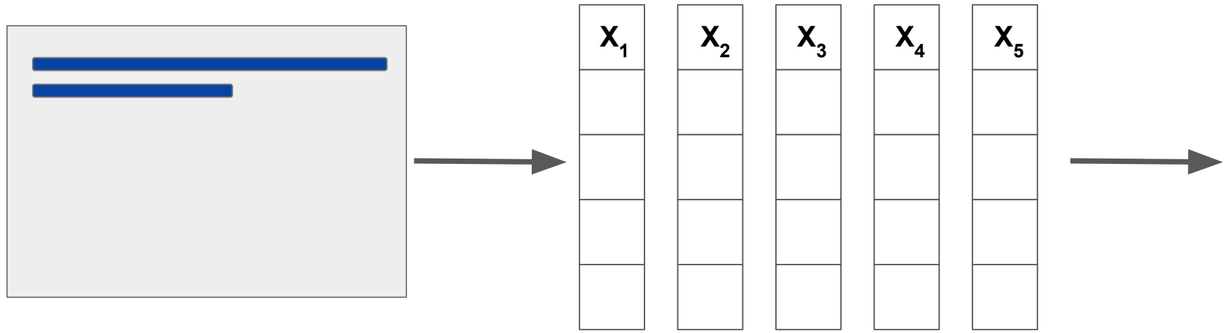
LANGCHAIN can be used to split large document into multiple smaller chunks so that RETRIEVER can combine it with input prompt and send it to the LLM

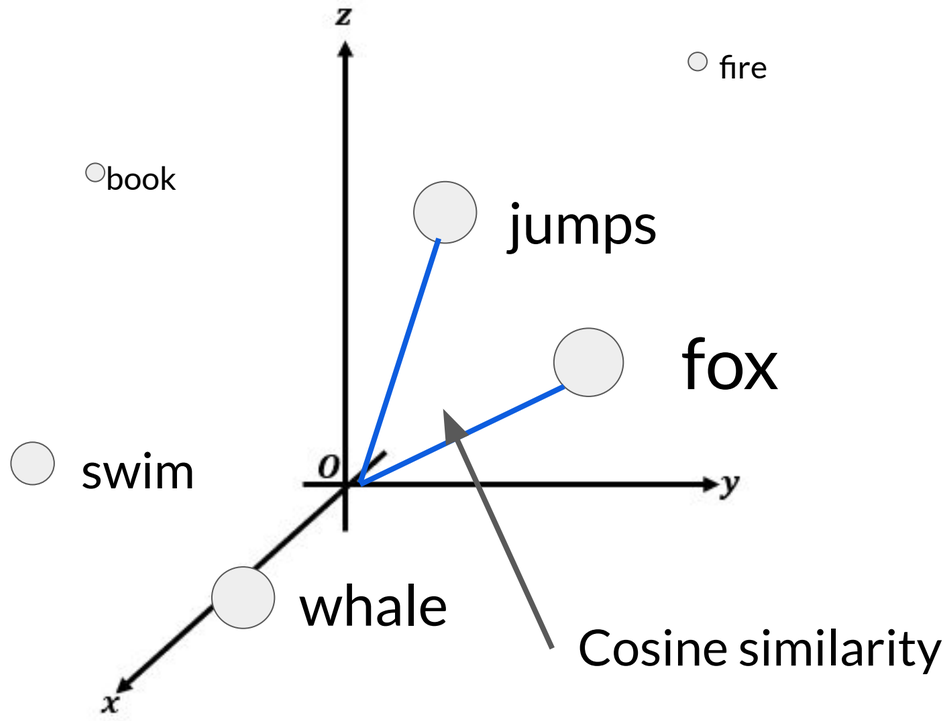
Data preparation for RAG

Two considerations for using external data in RAG:

1. Data must fit inside context window
2. Data must be in format that allows its relevance to be assessed at inference time: **Embedding vectors**

Prompt text converted to embedding vectors





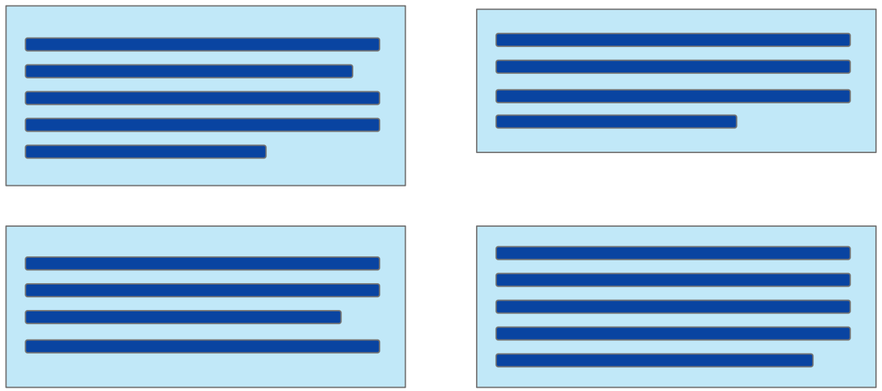
Each token(embeeded vectors) occupies a position in space(2D,3D etc) called vector embedding space which allow the LLM to identify semantically related words through cosine similarity measurement

Data preparation for RAG

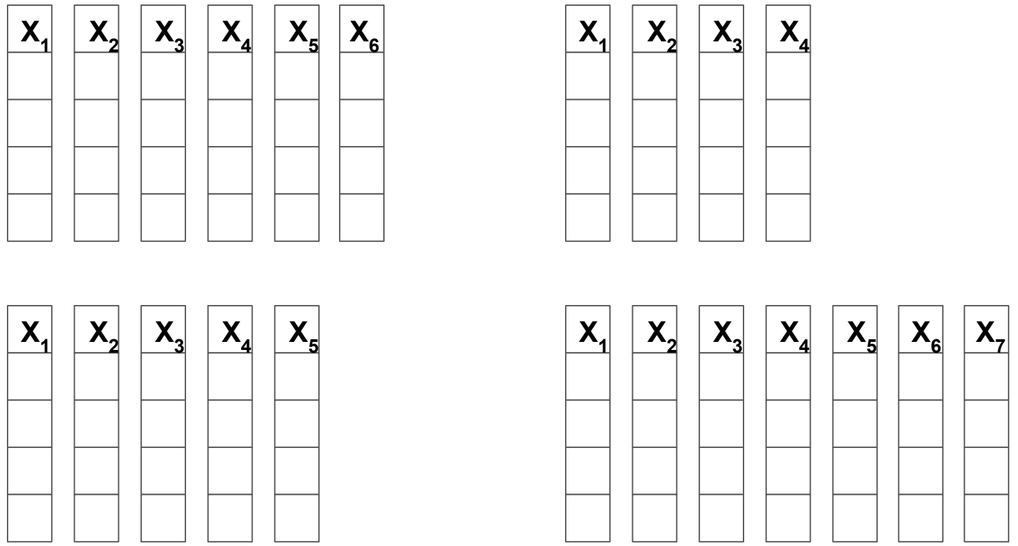
Two considerations for using external data in RAG:

1. Data must fit inside context window
2. Data must be in format that allows its relevance to be assessed at inference time: **Embedding vectors**

For External data source
Process each chunk with LLM
to produce embedding vectors

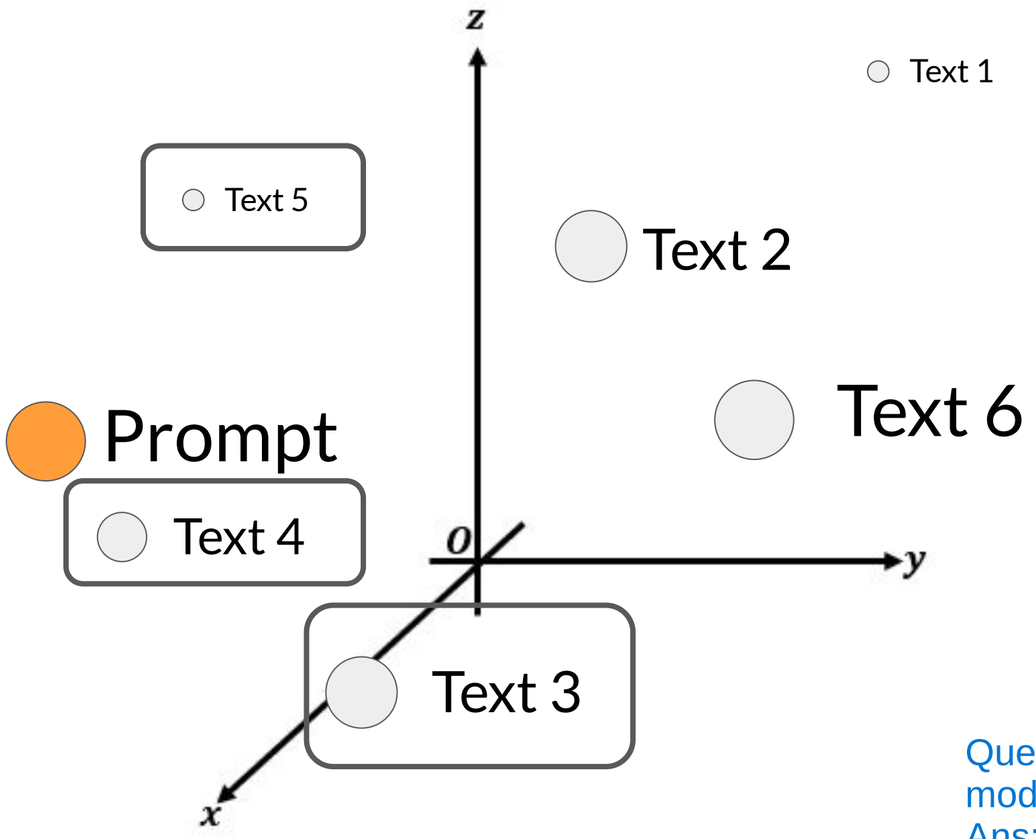


Each small chuck of external large data-source gets converted into respective small chunk of embedding vectors. These new representations of the data is stored in the structures called vector stores that:
- Enables faster searching of dataset
- Enables efficient identification of semantically related text



Vector database search

Vector databases are a particular implementation of a vector store where each vector is also identified by a key. This can allow, for instance, the text generated by RAG to also include a citation for the document from which it was received.



- Each text in vector store is identified by a key
- Enables a **citation** to be included in completion

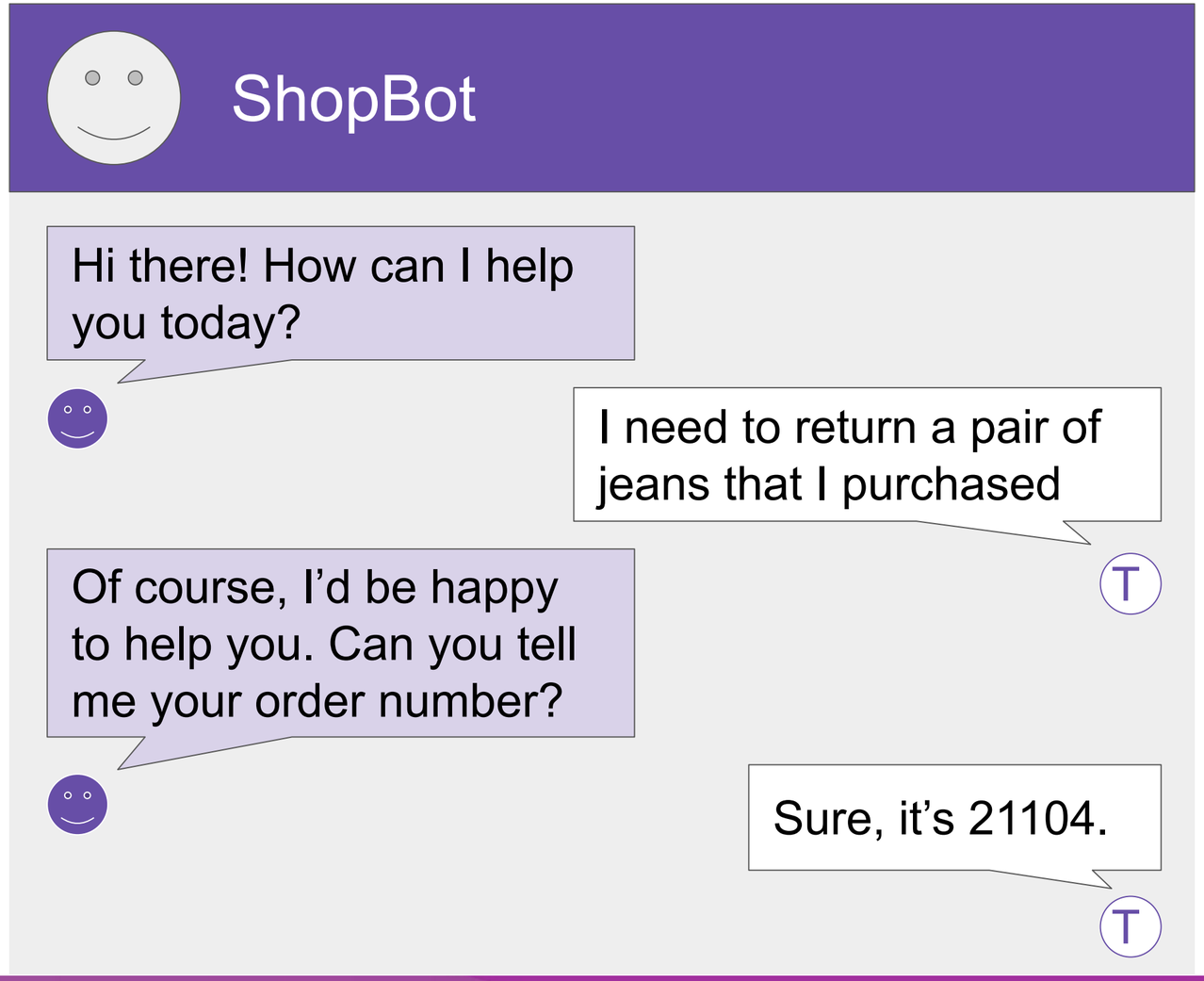
To summarize RAG can be used to interact with external data-sources and overcome the Knowledge cutoff and hallucinations issues.

Ques: How does Retrieval Augmented Generation (RAG) enhance generation-based models?
Ans: By making external knowledge available to the model

In Previous case we saw how LLMs using RAG can interact with EXTERNAL DATA-SOURCES. Now, in this section we will see that how LLMs can interact with the EXTERNAL APPLICATION (APIs)

Enabling interactions with external applications

Having an LLM initiate a clothing return



Having an LLM initiate a clothing return

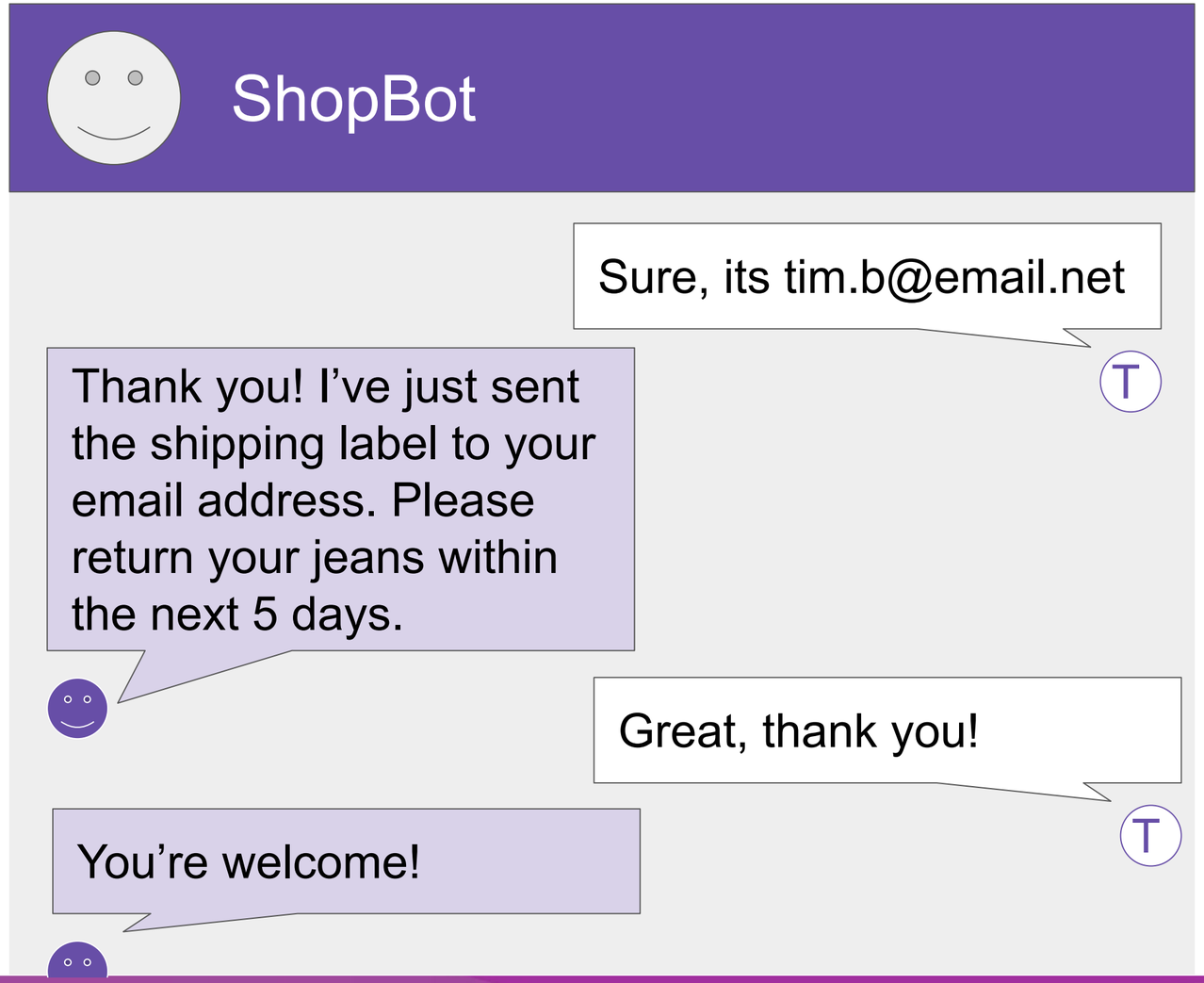
In this case here, you would likely be retrieving data through a SQL query to a back-end order database rather than retrieving data from a corpus of documents

Lookup with RAG

API call

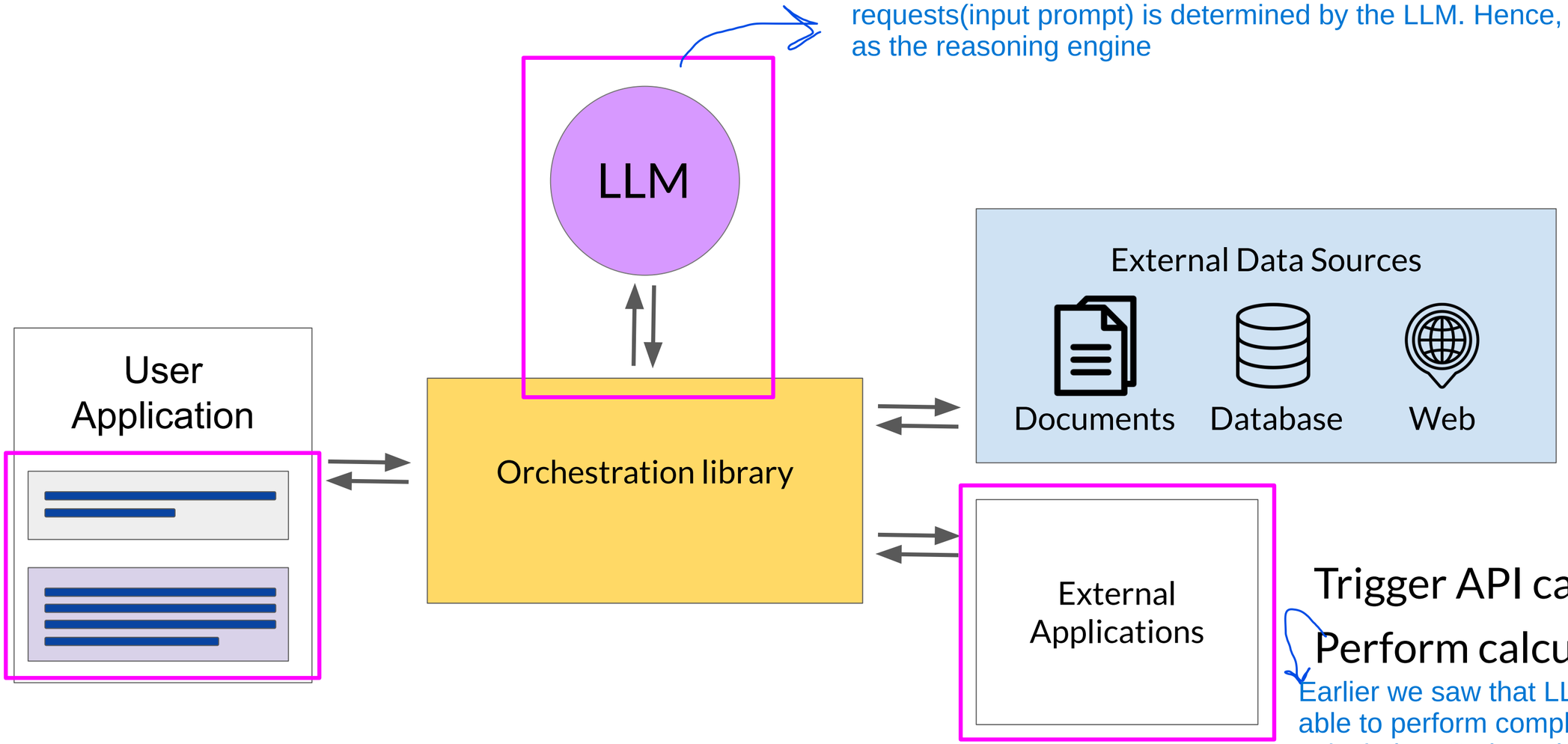
Having an LLM initiate a clothing return

API call to the shipper



LLM-powered applications

The actions that the app will take(completion) in response to user requests(input prompt) is determined by the LLM. Hence, LLMs acts as the reasoning engine



Trigger API call

Perform calculations

Earlier we saw that LLM was not able to perform complex calculation. So by using some external calcualtor API we can do the same

Requirements for using LLMs to power applications

Meaning that to trigger action, completions generated by the LLMs should contain below important information:

Plan actions

Steps to process return:

Step 1: Check order ID

Step 2: Request label

Step 3: Verify user email

Step 4: Email user label

First, the model needs to be able to generate a set of instructions so that the application knows what actions to take. These instructions need to be understandable and correspond to allowed actions

Format outputs

SQL Query:

```
SELECT COUNT(*)
FROM orders
WHERE order_id = 21104
```

Second, the completion needs to be formatted in a way that the broader application can understand. This could be as simple as a specific sentence structure or as complex as writing a script in Python or generating a SQL command

Validate actions

Collect required user information and make sure it is in the completion

User email:
tim.b@email.net

Lastly, the model may need to collect information that allows it to validate an action

Prompt structure is important!

As you saw, it is important that LLMs can reason through the steps that an application must take, to satisfy a user request. Unfortunately, complex reasoning can be challenging for LLMs, especially for problems that involve multiple steps or mathematics. These problems exist even in large models that show good performance at many other tasks

Helping LLMs reason and plan with Chain-of-Thought Prompting

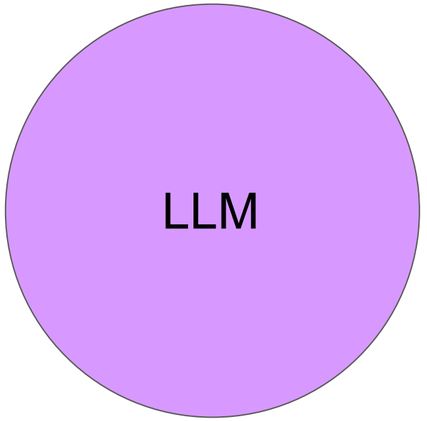
Chain on thought prompting approach can solve the above stated problem and help LLMs with complex reasoning

LLMs can struggle with complex reasoning problems

Prompt

```
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?
```

Model



Here we are tuning our LLM using one shot inferencing followed by giving a input prompt having complex reasoning. We see the one shot inferencing for such complex prompt is not useful since it is generating the wrong answer

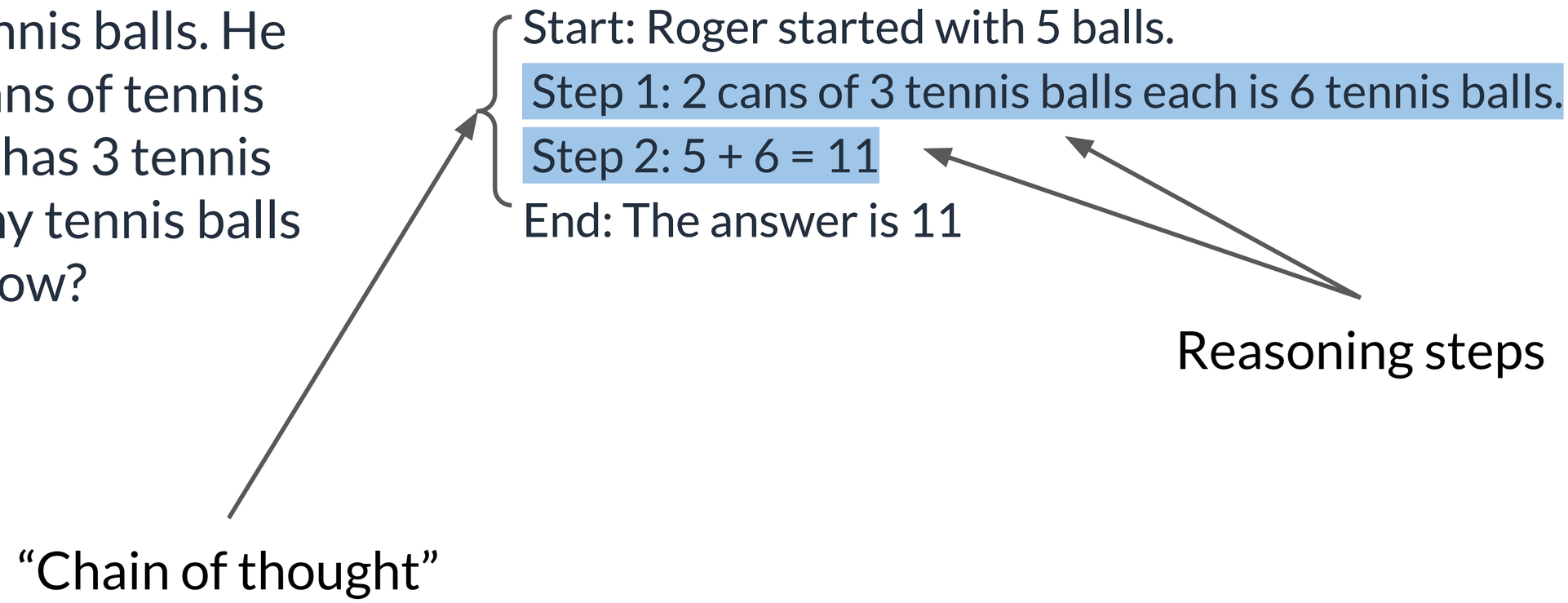
Completion

```
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

A: The answer is 27.
```

Humans take a step-by-step approach to solving complex problems

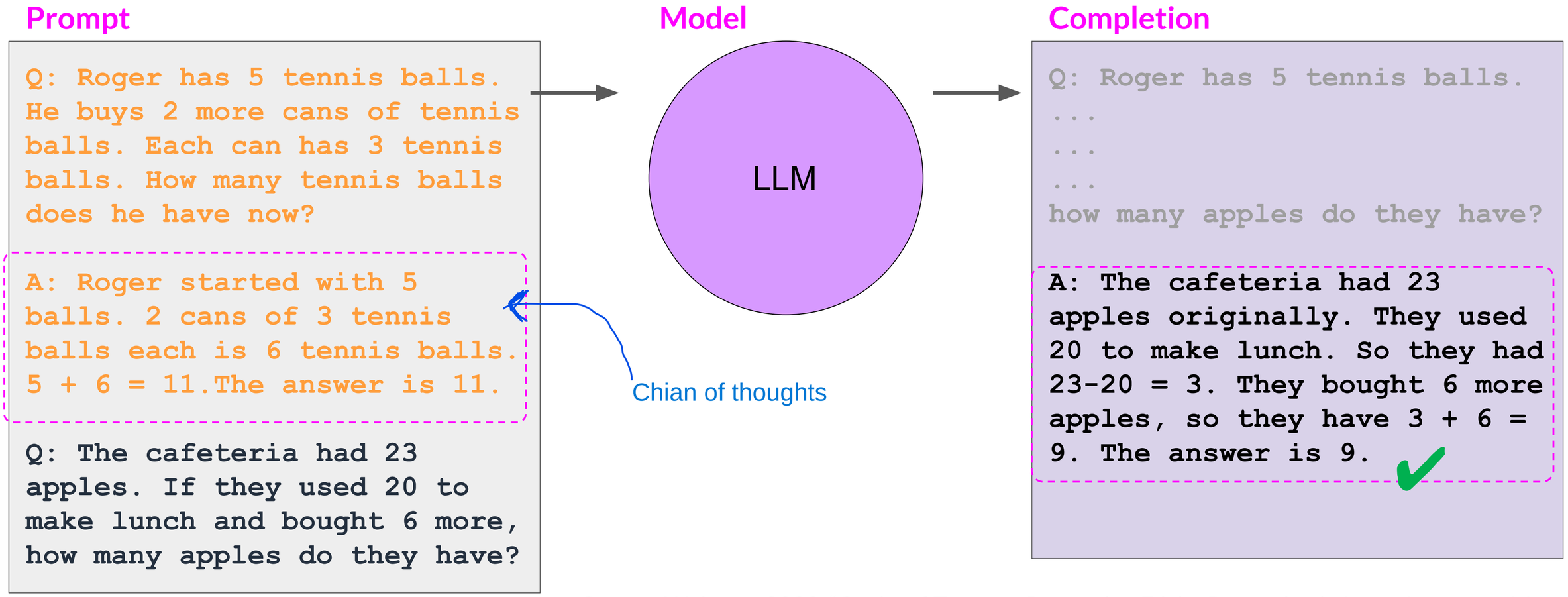
In this slide we will see that how a human will approach to this problem statement. A human will use divide and conquer approach in which he/she will be dividing the problem into smaller chunks that collectively is known as chain of thoughts

Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?



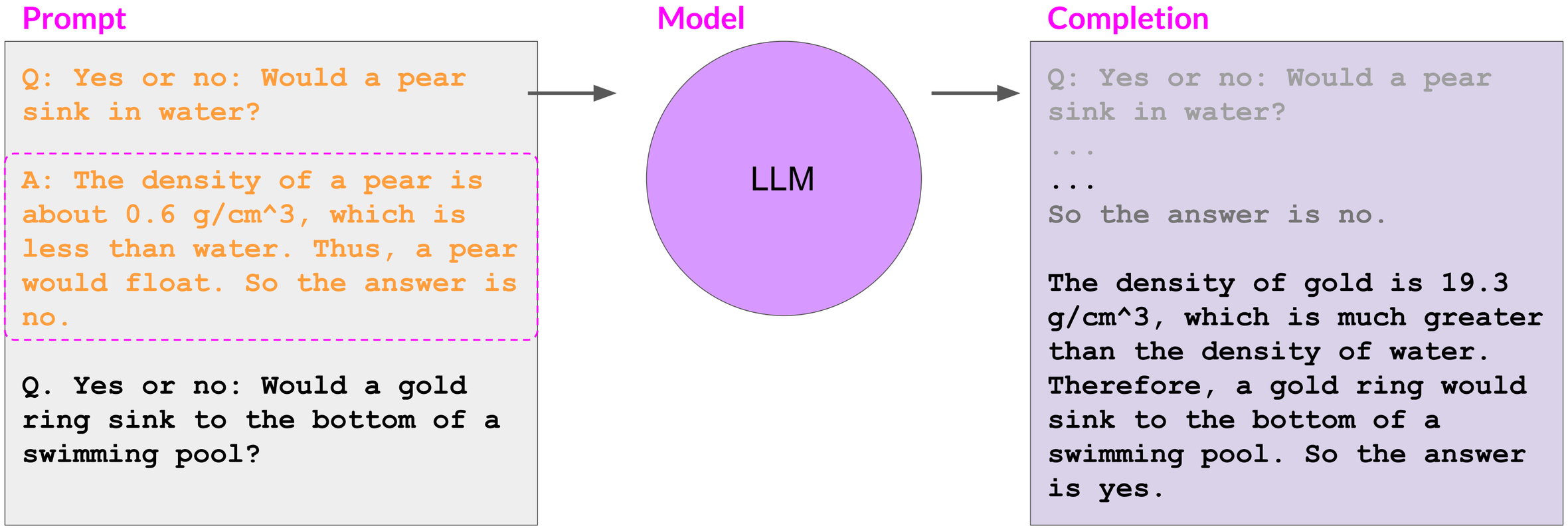
Chain-of-Thought Prompting can help LLMs reason

Here we can see that when in the same example when we modified the input prompt with chain of thoughts we start getting the correct output



Source: Wei et al. 2022, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”

Chain-of-Thought Prompting can help LLMs reason



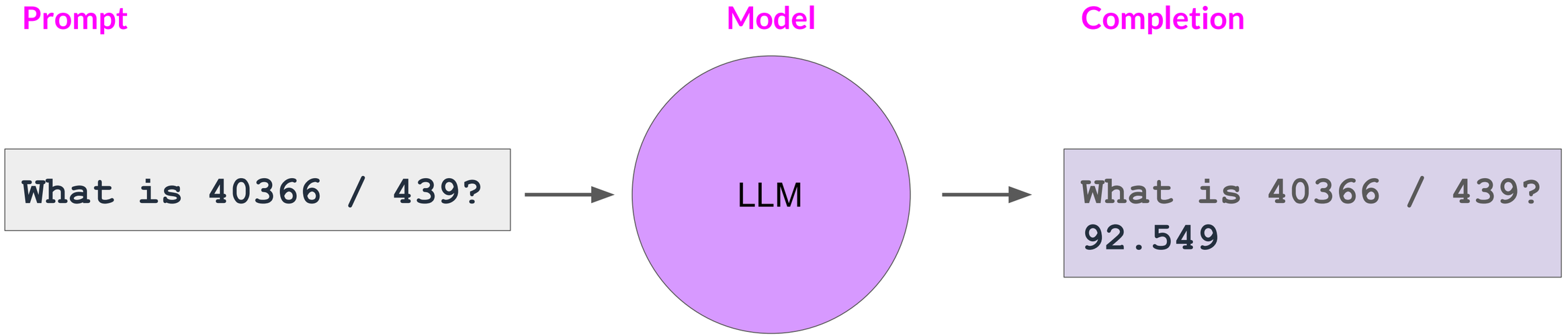
LLM knows about density of gold since it is trinaed to remember that dutring training stage.
Also, LLM generates swimming pool and water as closely realted texts. Since in vector embedding space these 2 texts will occupy position who distance with each other will be less

Source: Wei et al. 2022, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”

In above slide we have seen how LLM can reason through complex reasoning using CHIAN OF THOUGHTS. Now in following slide we will see that how by using PROGRAM-AIDED LANGUAGE MODELS(PAL) LLMs will be able to do the complex calculations(Simple calculation like addition LLM was able to do using chain of thoughts but for complex ones PAL would be needed)

Program-aided Language Models

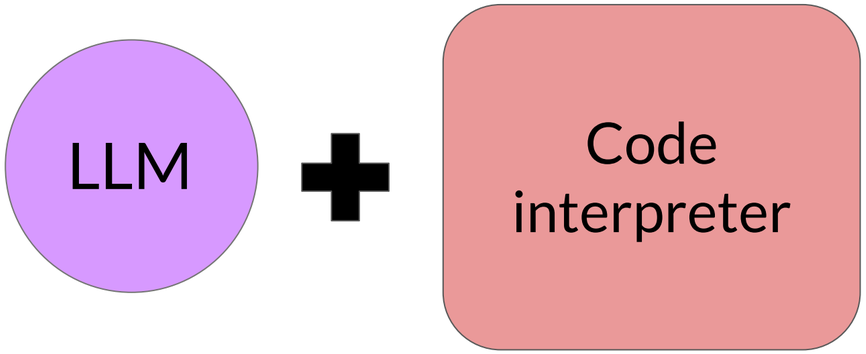
LLMs can struggle with mathematics



Here LLM is giving calculated result. Please note here we cannot solely blame the LLM since LLM are not meant for calculation. Here in this example LLM is simply giving the next most probable token that complete the input prompt generating the Output/Completion

Program-aided language (PAL) models

We can overcome the above mentioned issue by making LLM model to interact with externa application like Python Interpreter to carry out the complex calculations



PAL used chain of thought to generate the executable python script which is then passed to the interpreter to execute

The strategy behind PAL is to have the LLM generate completions where reasoning steps are accompanied by computer code. This code is then passed to an interpreter to carry out the calculations necessary to solve the problem. You specify the output format for the model by including examples for one or few short inference in the prompt

Source: Gao et al. 2022, “PAL: Program-aided Language Models”

Chain-of-Thought (Wei et al., 2022)

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 tennis balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

Model Output

A: The bakers started with 200 loaves. They sold 93 in the morning and 39 in the afternoon. So they sold 93 + 39 = 132 loaves. The grocery store returned 6 loaves. So they had 200 - 132 - 6 = 62 loaves left.
The answer is 62.

Program-aided Language models (this work)

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 tennis balls.
```
tennis_balls = 5
```
2 cans of 3 tennis balls each is
```
bought_balls = 2 * 3
```
tennis balls. The answer is
```
answer = tennis_balls + bought_balls
```

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

Model Output

A: The bakers started with 200 loaves
```
loaves_baked = 200
```
They sold 93 in the morning and 39 in the afternoon
```
loaves_sold_morning = 93
loaves_sold_afternoon = 39
```
The grocery store returned 6 loaves.
```
loaves_returned = 6
```
The answer is
```
answer = loaves_baked - loaves_sold_morning
  - loaves_sold_afternoon + loaves_returned
```
```
>>> print(answer)
74
```

PAL example

Prompt with one-shot example

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

Answer:

```
# Roger started with 5 tennis balls
tennis_balls = 5
# 2 cans of tennis balls each is
bought_balls = 2 * 3
# tennis balls. The answer is
answer = tennis_balls + bought_balls
```

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves did they have left?

Steps highlighted in blue represents the reasoning steps that is achieved using the chain of thoughts

Steps highlighted in pink translates any reasoning step that involve calculation into code(pythonic code)

Notice that in the Prompt example each Reasoning step starts with the # sign which signifies then Python interpreter will treat them as comment and skip. Which means interpreter will only execute the pink highlighted text obtained from chain of thoughts over a external python interpreter using PAL

PAL example

Prompt with one-shot example

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

Answer:

```
# Roger started with 5 tennis balls
tennis_balls = 5
# 2 cans of tennis balls each is
bought_balls = 2 * 3
# tennis balls. The answer is
answer = tennis_balls + bought_balls
```

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves did they have left?

Completion, CoT reasoning (blue) , and PAL execution (pink)

Answer:

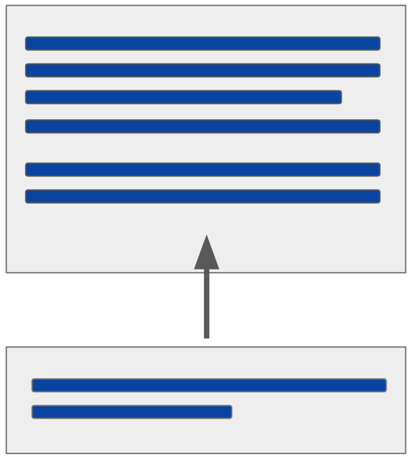
```
# The bakers started with 200 loaves
loaves_baked = 200
# They sold 93 in the morning and 39 in the
afternoon
loaves_sold_morning = 93
loaves_sold_afternoon = 39
# The grocery store returned 6 loaves.
loaves_returned = 6
# The answer is
answer = loaves_baked
  - loaves_sold_morning
  - loaves_sold_afternoon
  + loaves_returned
```

Observe here that models understands automatically that which term needs to be added or subtracted

Program-aided language (PAL) models

Includes input instruction prompt and one shot/few shot inferences which are arranged in chain of thoughts and respective python representation as shown in earlier slides.

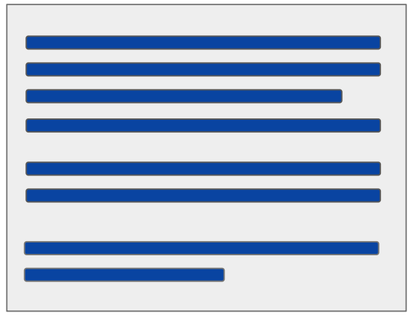
PAL prompt template



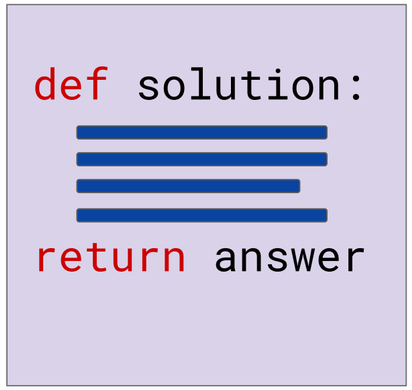
question

New input instruction that needs to be out by LLM that involves complex reasoning and complex calculations

LLM



PAL formatted prompt

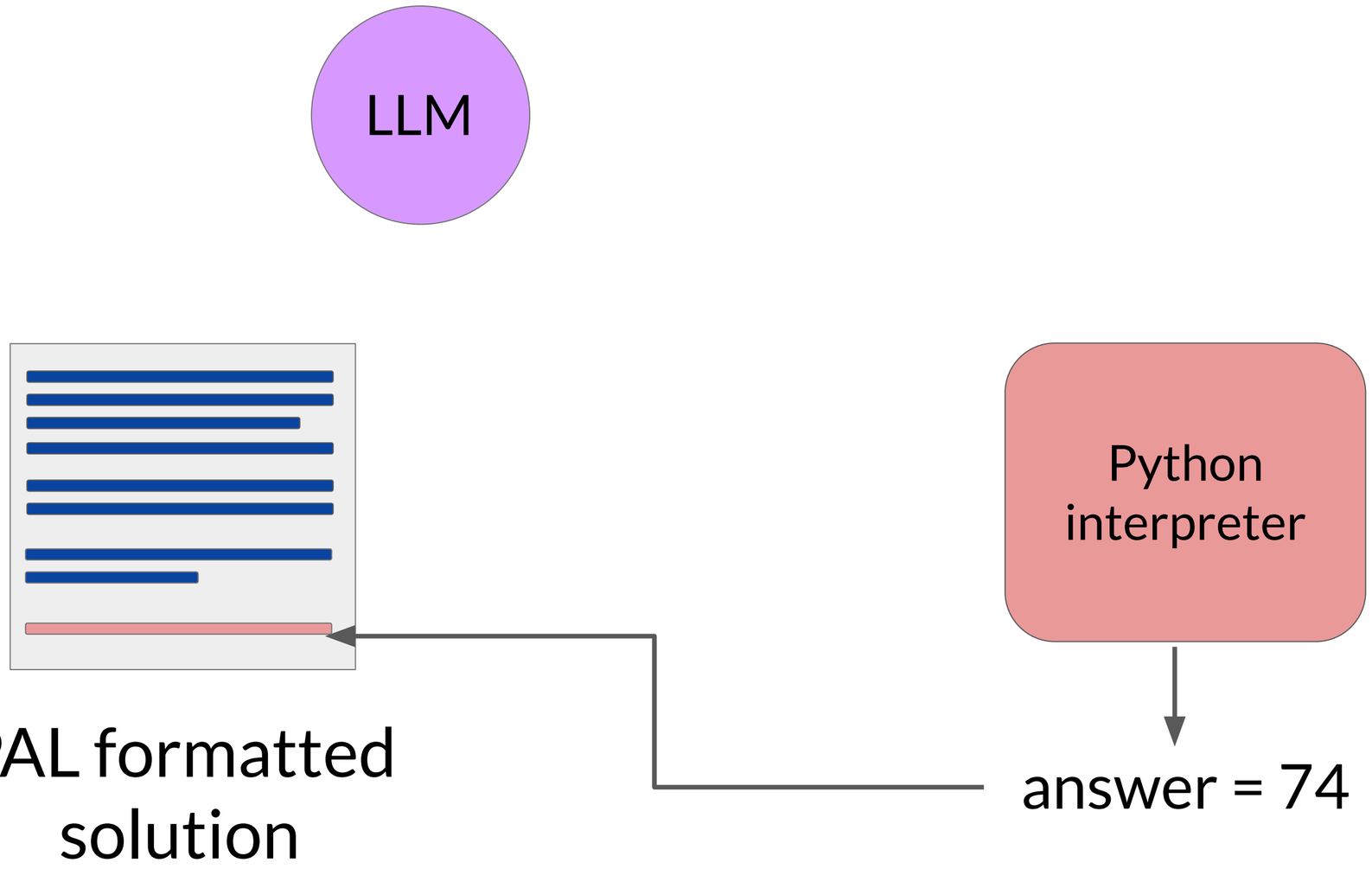


Python script

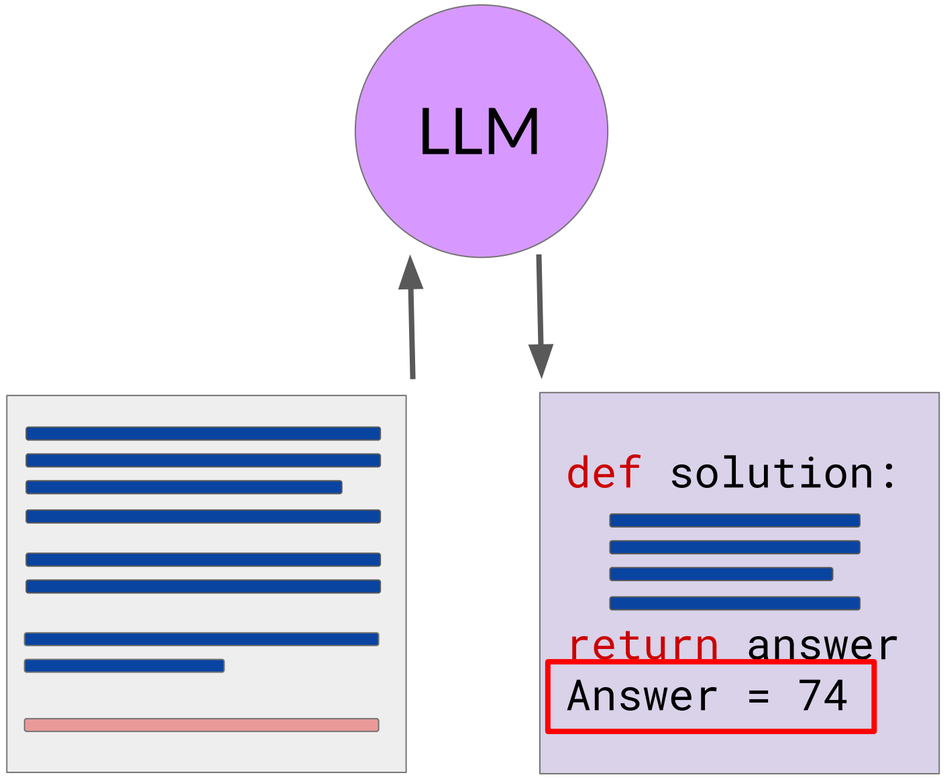
Python interpreter

answer = 74

Program-aided language (PAL) models



Program-aided language (PAL) models

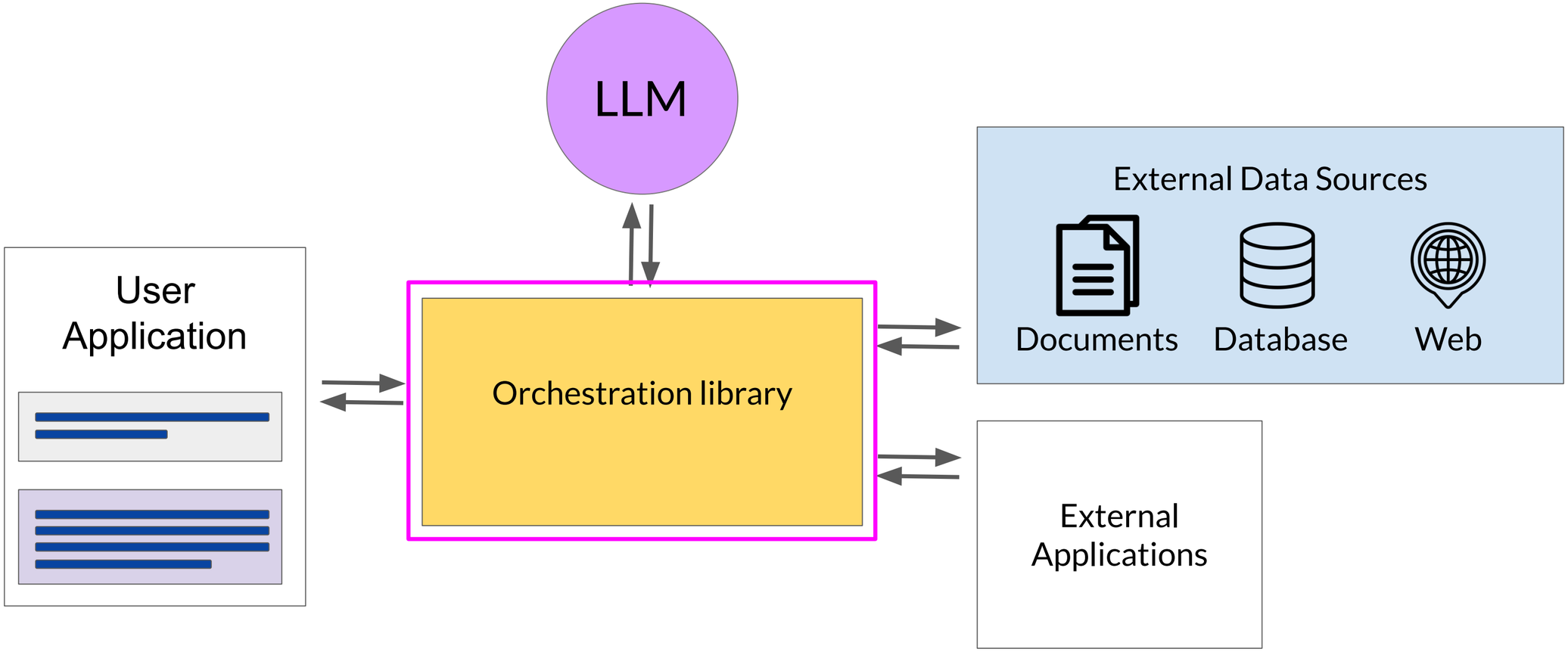


PAL formatted solution

Completion with correct answer

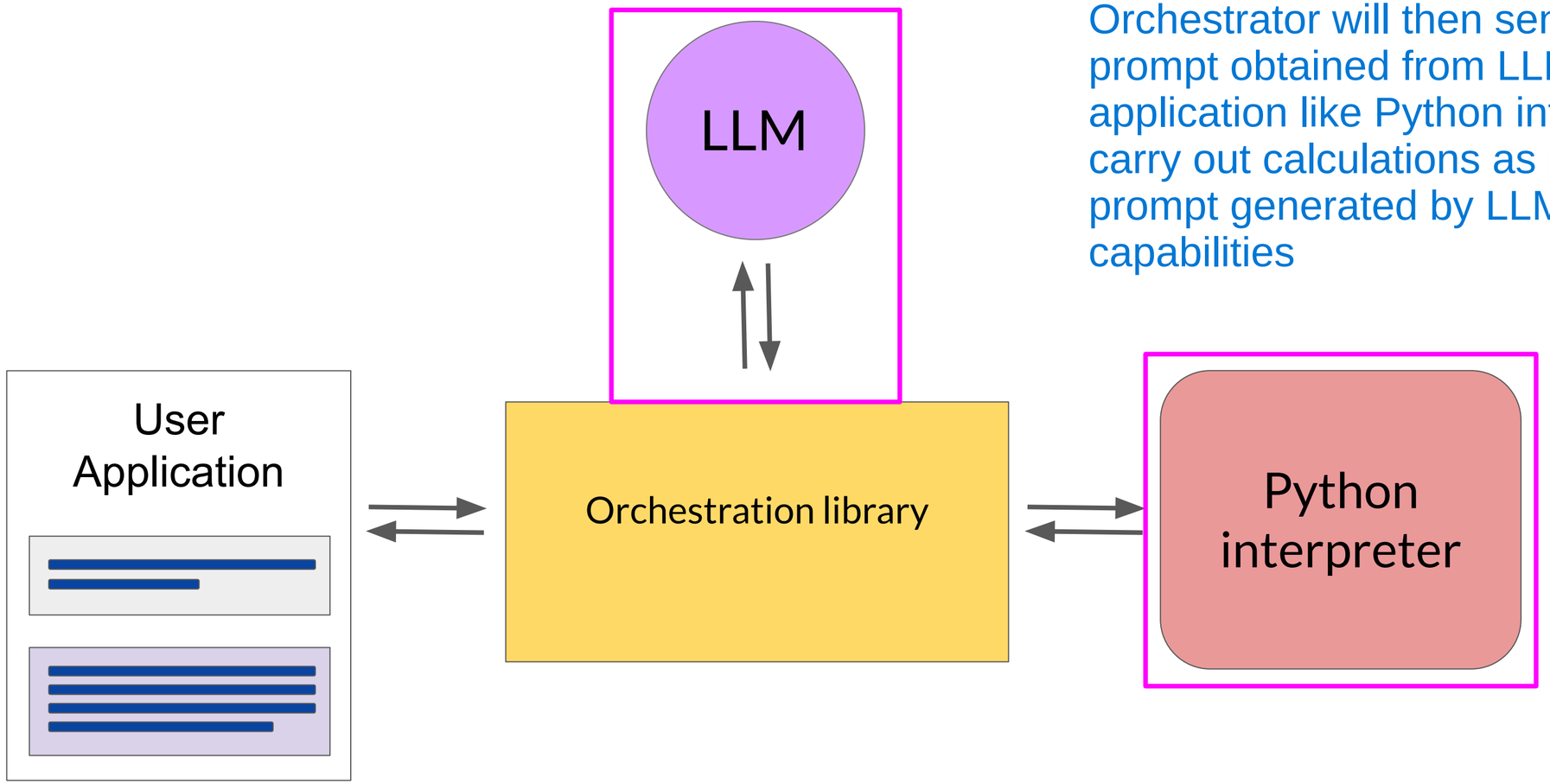
LLM-powered applications

A question can come into our mind that how can we automate this process of so we don't need to pass the information back and forth between LLM and interpreter by hand(manually). Orchestration library can be utilized for the same

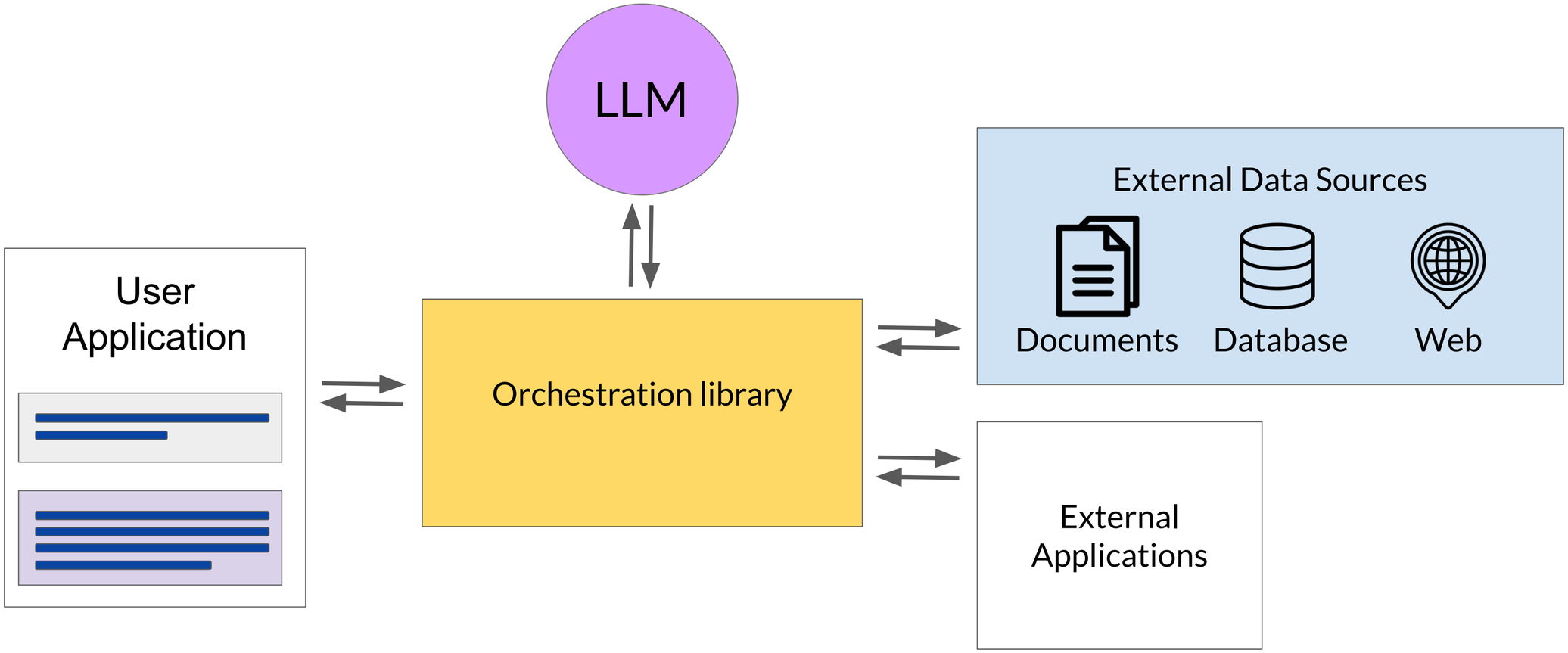


PAL architecture

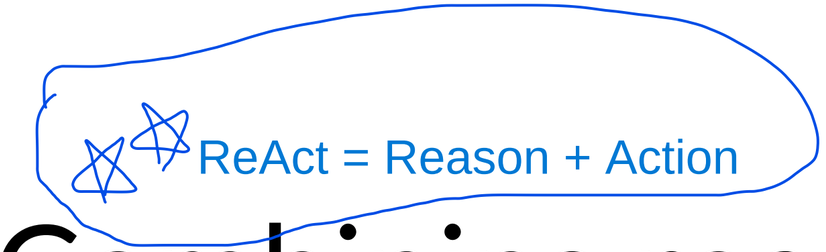
Note that LLM is still action as a reasoning engine (generating chain of thoughts in the form of reasoning and respective python code-->Input Prompt). Orchestrator will then send this input prompt obtained from LLM to External application like Python interpreter to carry out calculations as present in the prompt generated by LLMs resoning capabilities



LLM-powered applications



In Shop bot example we saw that needs LLM can Plan actions, Produce Output and Validate actions. In PAL saw how to carry out complex calculation. In real world scenario LLM are very complex and not as easy that we made to understand these concepts. Now we will see how basics learned so far can be applied to create or a more complex LLM with more complex reasoning and complex calculations

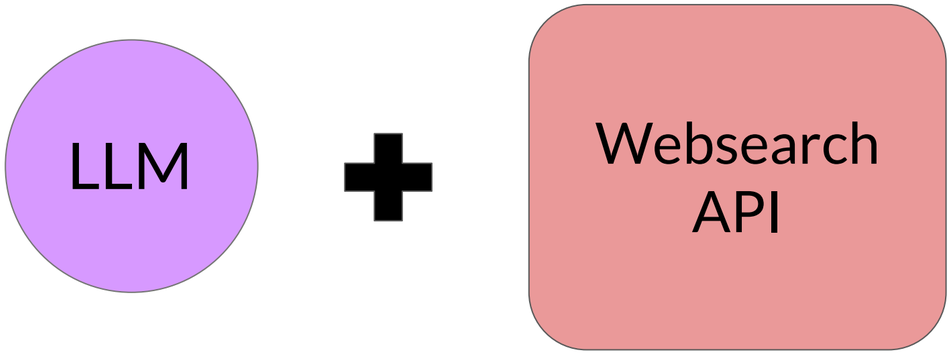


ReAct: Combining reasoning and action in LLMs

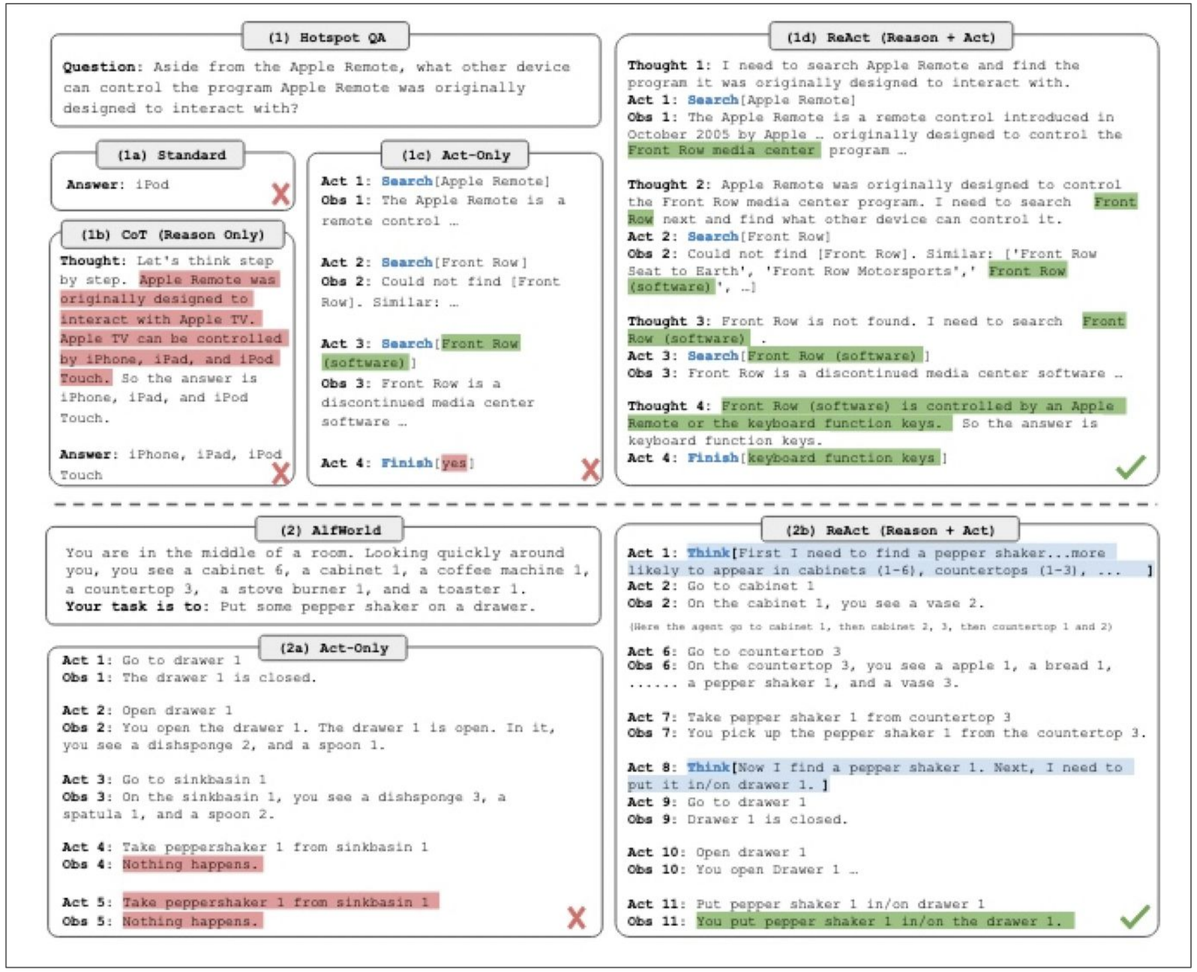
Most applications will require the LLM to manage more complex workflows, perhaps in including interactions with multiple external data sources and applications. In this video, you'll explore a framework called ReAct that can help LLMs plan out and execute these workflows. ReAct is a prompting strategy that combines chain of thought reasoning with action planning

ReAct: Synergizing Reasoning and Action in LLMs

- This paper was proposed by researchers at Princeton and Google in 2022
- The paper develops a series of complex prompting examples based on problems from Hot Pot QA, a multi-step question answering benchmark that requires reasoning over two or more Wikipedia passages and fever, a benchmark that uses Wikipedia passages to verify facts

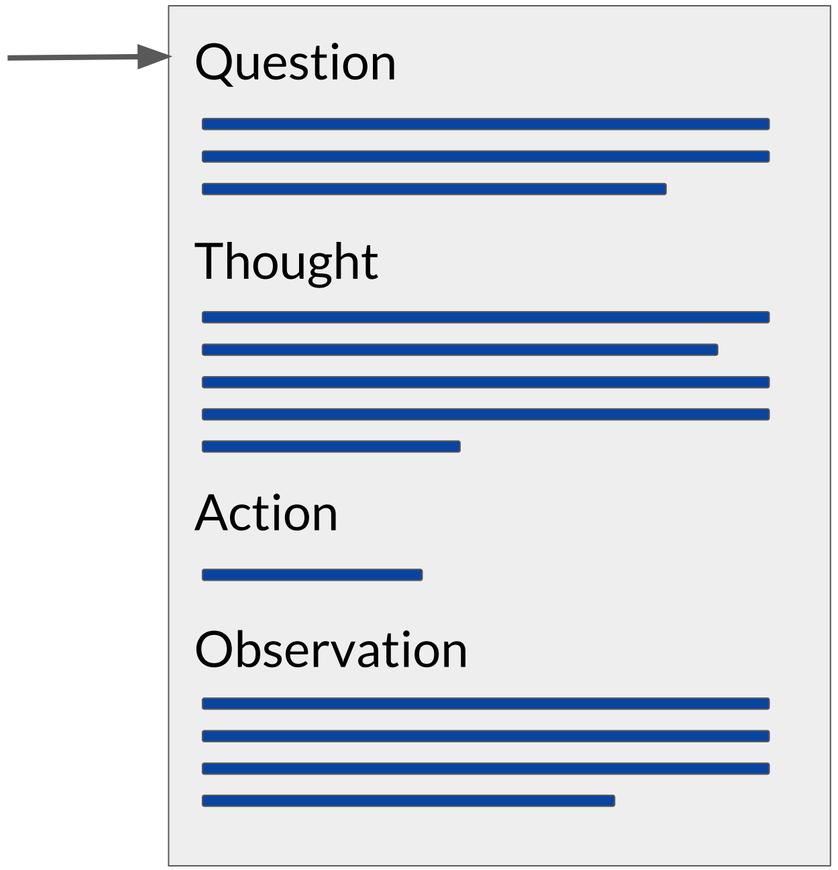


HotPot QA: multi-step question answering
Fever: Fact verification



Source: Yao et al. 2022, “ReAct: Synergizing Reasoning and Acting in Language Models”

ReAct: Synergizing Reasoning and Action in LLMs



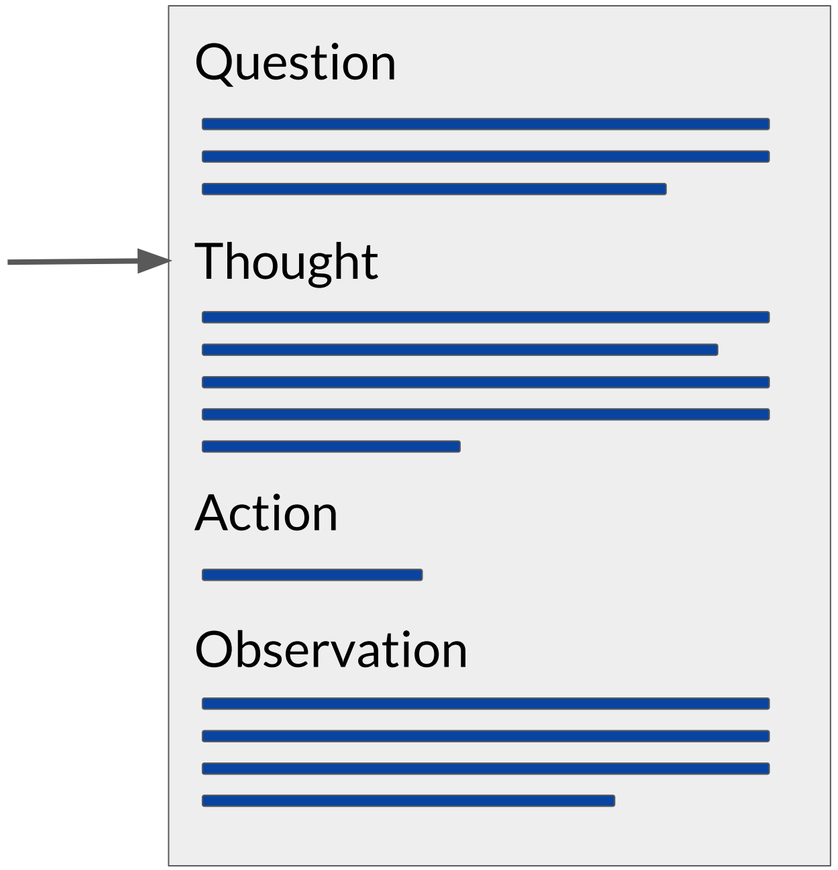
Question: Problem that requires advanced reasoning and multiple steps to solve.

E.g.

"Which magazine was started first, *Arthur's Magazine* or *First for Women*?"

Source: Yao et al. 2022, "ReAct: Synergizing Reasoning and Acting in Language Models"

ReAct: Synergizing Reasoning and Action in LLMs

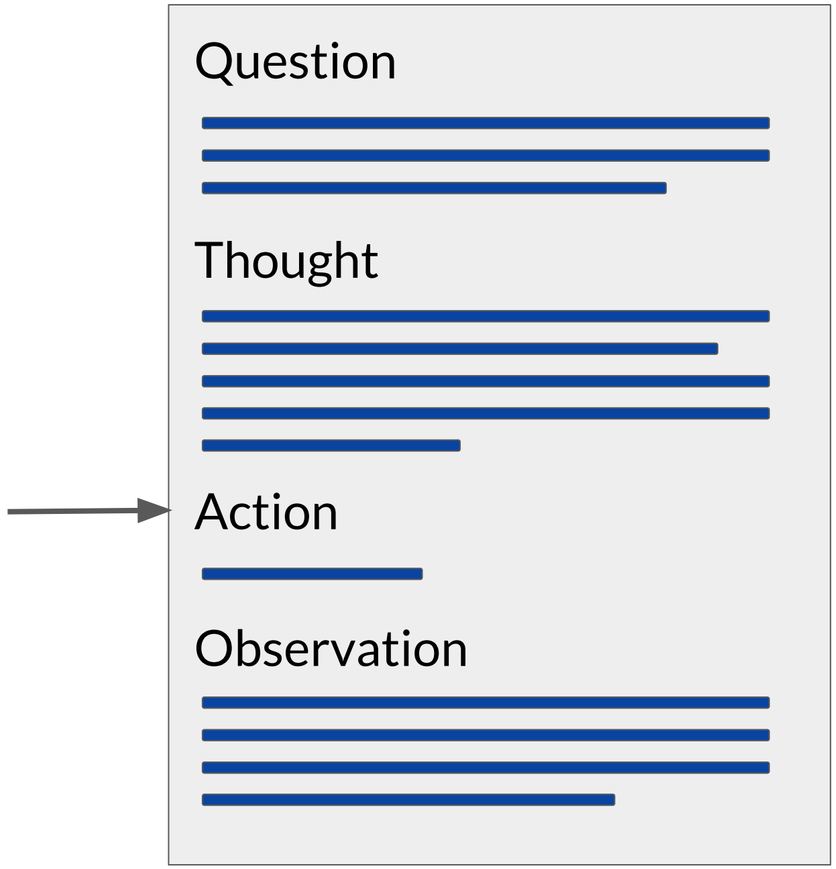


Thought: A reasoning step that identifies how the model will tackle the problem and identify an action to take.

“I need to search Arthur’s Magazine and First for Women, and find which one was started first.”

ReAct: Synergizing Reasoning and Action in LLMs

In the case of the ReAct framework, the authors created a small Python API to interact with Wikipedia. The three allowed actions are search, which looks for a Wikipedia entry about a particular topic lookup, which searches for a string on a Wikipedia page and finish, which the model carries out when it decides it has determined the answer



Action: An external task that the model can carry out from an allowed set of actions.

E.g.

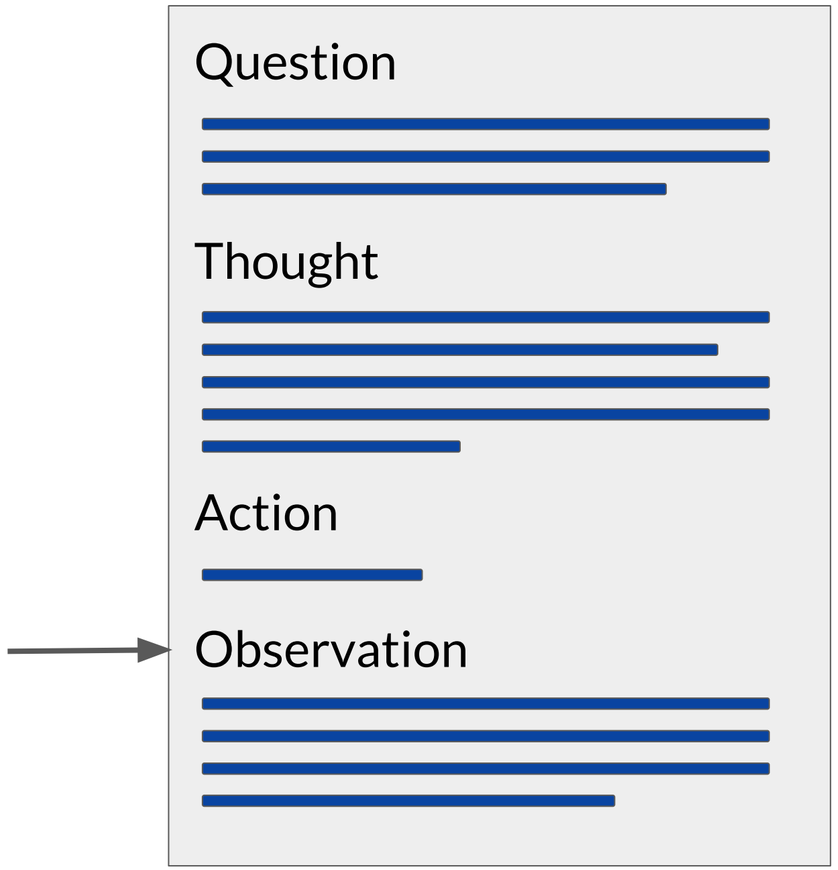
```
search[entity]
lookup[string]
finish[answer]
```

action to be taken formatted by passing them within the Square Brackets [ ]

Which one to choose is determined by the information in the preceding thought.

```
search[Arthur's Magazine]
```

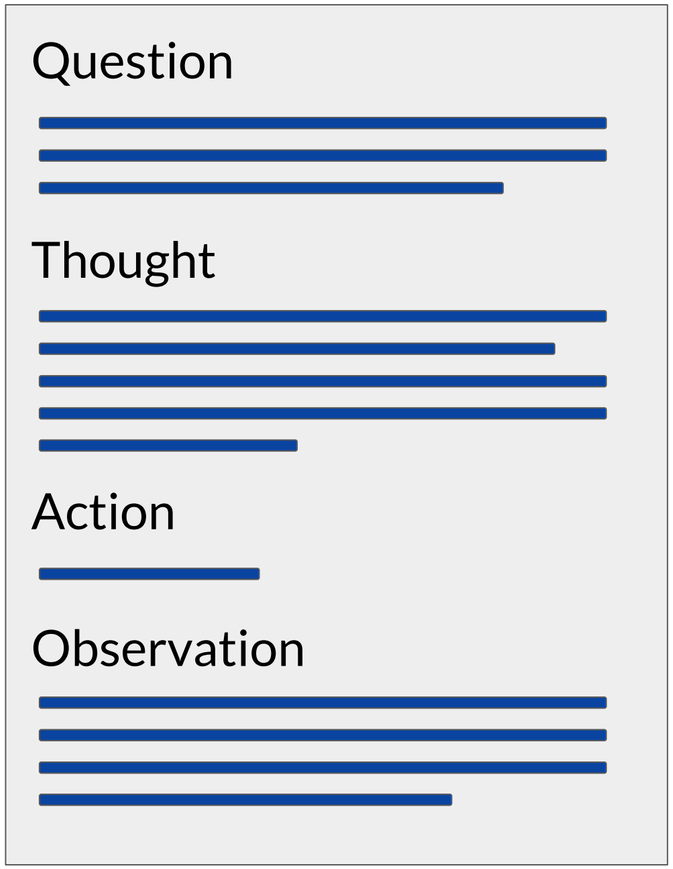
ReAct: Synergizing Reasoning and Action in LLMs



Observation: the result of carrying out the action

E.g.
“Arthur’s Magazine (1844-1846) was an American literary periodical published in Philadelphia in the 19th century.”

ReAct: Synergizing Reasoning and Action in LLMs



The model will iterate over these 3 trios multiple times until it produces the desired outcome

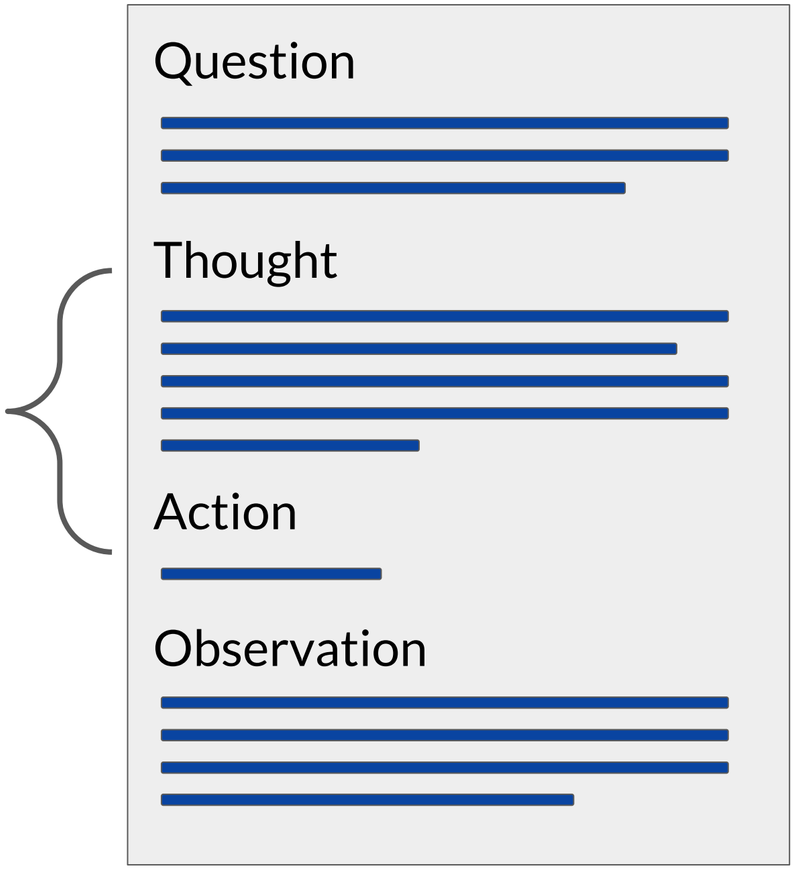
Seconds iteration

Thought 2:
"Arthur's magazine was started in 1844. I need to search First for Women next."

Action 2:
`search[First for Women]`

Observation 2:
"First for Women is a woman's magazine published by Bauer Media Group in the USA.[1] The magazine was started in 1989."

ReAct: Synergizing Reasoning and Action in LLMs



Third iteration

Thought 3:

“First for Women was started in 1989. 1844 (Arthur’s Magazine) < 1989 (First for Women), so Arthur’s Magazine as started first”

Action 2:

`finish``[Arthur’s Magazine]`

At this stage we have out desired result that is Arthur's Magazine was published first. There in this step called finished action where we are basically returning the final outcome that LLM model has generated using ReAct framework

ReAct instructions define the action space

```
Solve a question answering task with interleaving Thought, Action,
Observation steps.

Thought can reason about the current situation, and Action can be
three types:
(1) Search[entity], which searches the exact entity on Wikipedia and
returns the first paragraph if it exists. If not, it will return
some similar entities to search.
(2) Lookup[keyword], which returns the next sentence containing
keyword in the current passage.
(3) Finish[answer], which returns the answer and finishes the task.
Here are some examples.
```

The final sentence in the instructions lets the LLM know that some examples will come next in the prompt text

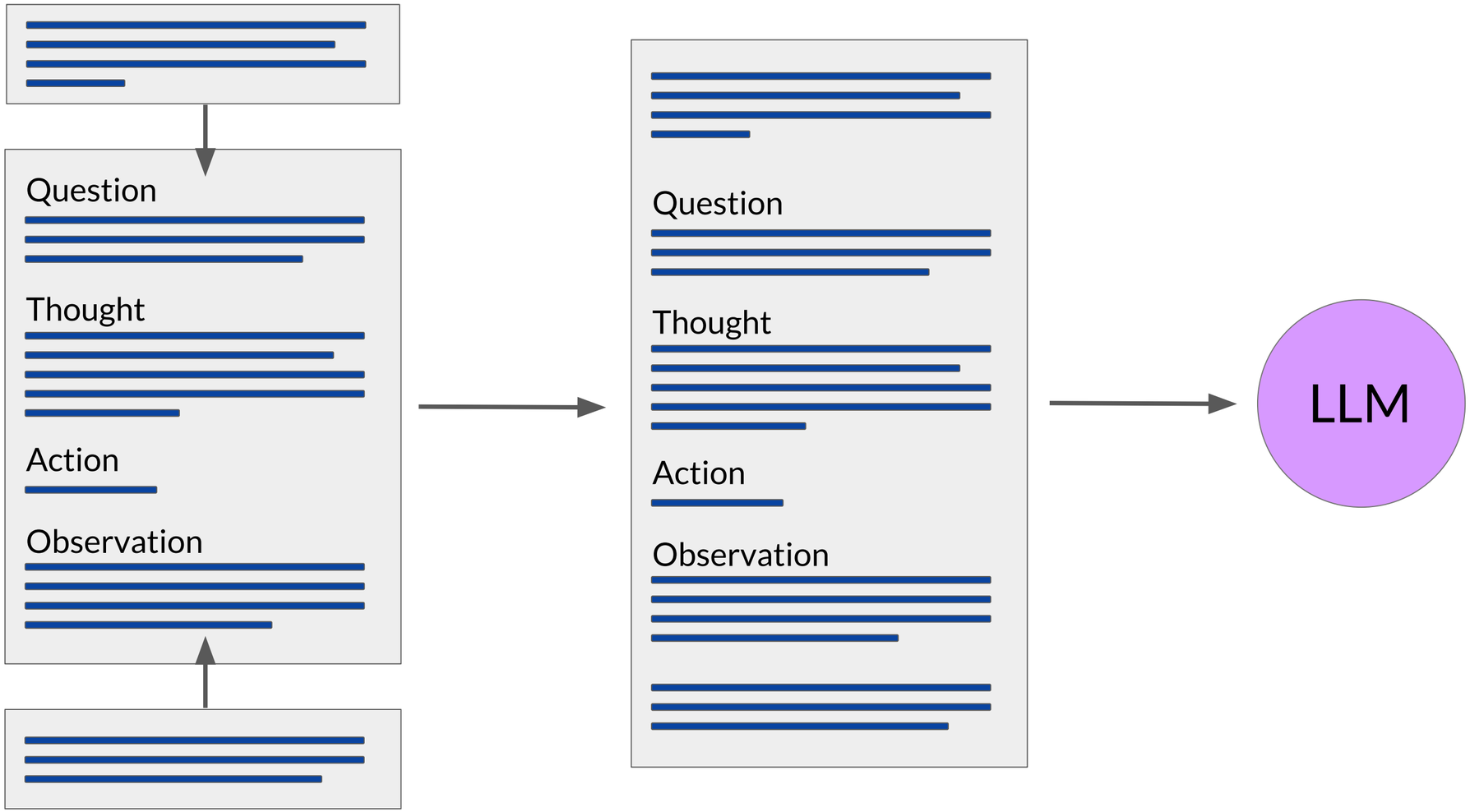
Building up the ReAct prompt

Instructions

ReAct example

(could be more than one example)

Question to be answered

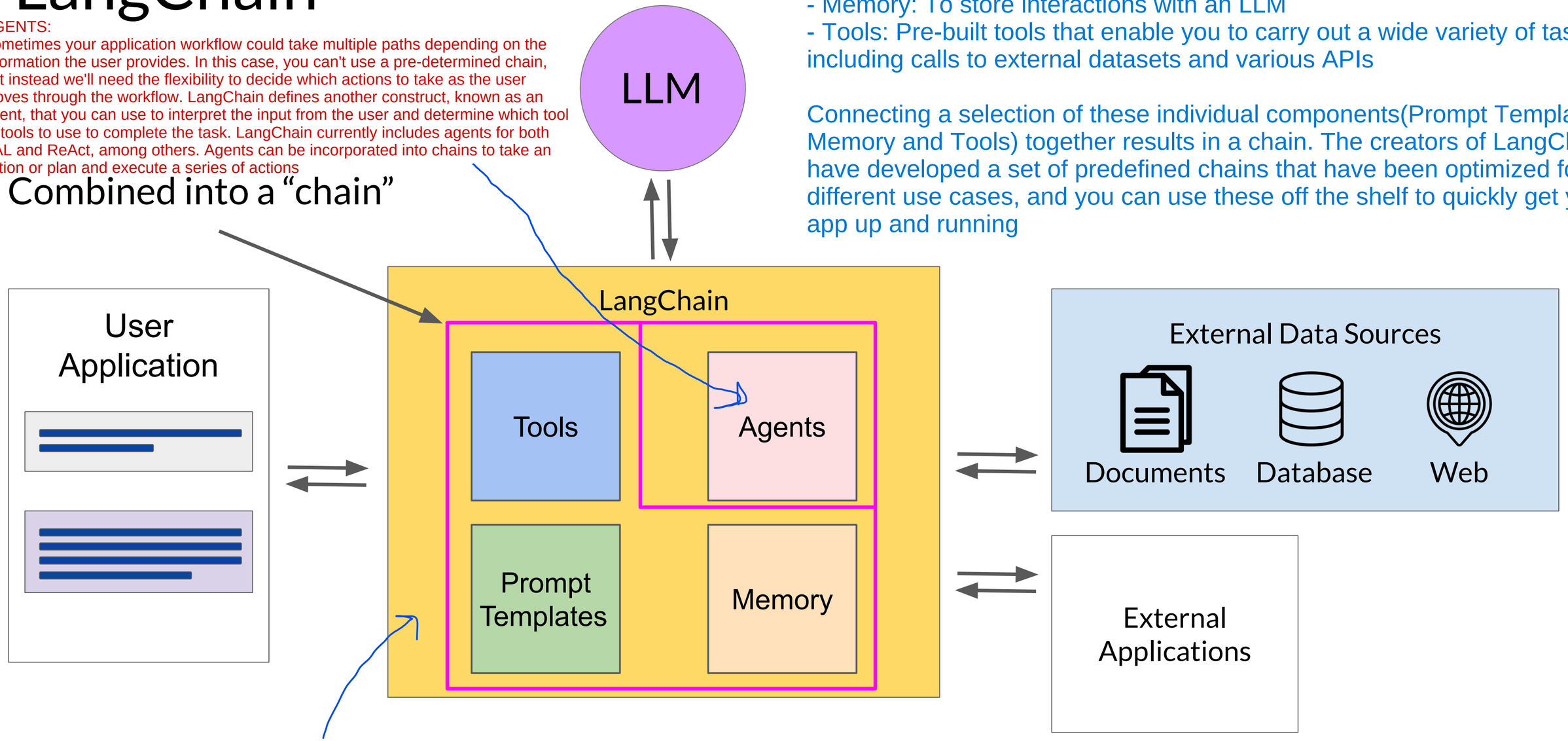


LangChain

AGENTS:
Sometimes your application workflow could take multiple paths depending on the information the user provides. In this case, you can't use a pre-determined chain, but instead we'll need the flexibility to decide which actions to take as the user moves through the workflow. LangChain defines another construct, known as an agent, that you can use to interpret the input from the user and determine which tool or tools to use to complete the task. LangChain currently includes agents for both PAL and ReAct, among others. Agents can be incorporated into chains to take an action or plan and execute a series of actions

- Prompt Templates: for creating many templates for different use cases that you can use to format both input examples and model completions
- Memory: To store interactions with an LLM
- Tools: Pre-built tools that enable you to carry out a wide variety of tasks, including calls to external datasets and various APIs

Connecting a selection of these individual components(Prompt Templates, Memory and Tools) together results in a chain. The creators of LangChain have developed a set of predefined chains that have been optimized for different use cases, and you can use these off the shelf to quickly get your app up and running



Note this big yellow box is representing Orchestration library. So we can say that Langchainor LanguageChain is the language framework for LLM that works as an orchestration library

The significance of scale: application building

ReAct=Reason+Action.

Abhishek Keasri ko samjhana ki instead of taking action directly first reason you action and if found approproate then take action. Eg; He was inspecting some element on the webpage which highlighted some span tag and he directly took the xpath. I explained him ki atleast check toh karo ki some ID is available or not if not check classname present is unique or not, and also than span tag was under tr tag so ideally Abhishek required to take xpath of tr and not span. So, this is how I embeded Abhishek with Reason and Action

BERT*
110M

BLOOM
176B

*Bert-base

The last thing to keep in mind as you develop applications using LLMs is that the ability of the model to reason well and plan actions depends on its scale. Larger models are generally your best choice for techniques that use advanced prompting, like PAL or ReAct. Smaller models may struggle to understand the tasks in highly structured prompts and may require you to perform additional fine tuning to improve their ability to reason and plan

Read this reading from the course page "ReAct: Reasoning and action". Here comparisions between four prompting methods: Standard, Chain-of-thought (CoT, Reason Only), Act-only, and ReAct (Reason+Act) for solving a HotpotQA question

LLM powered application architectures

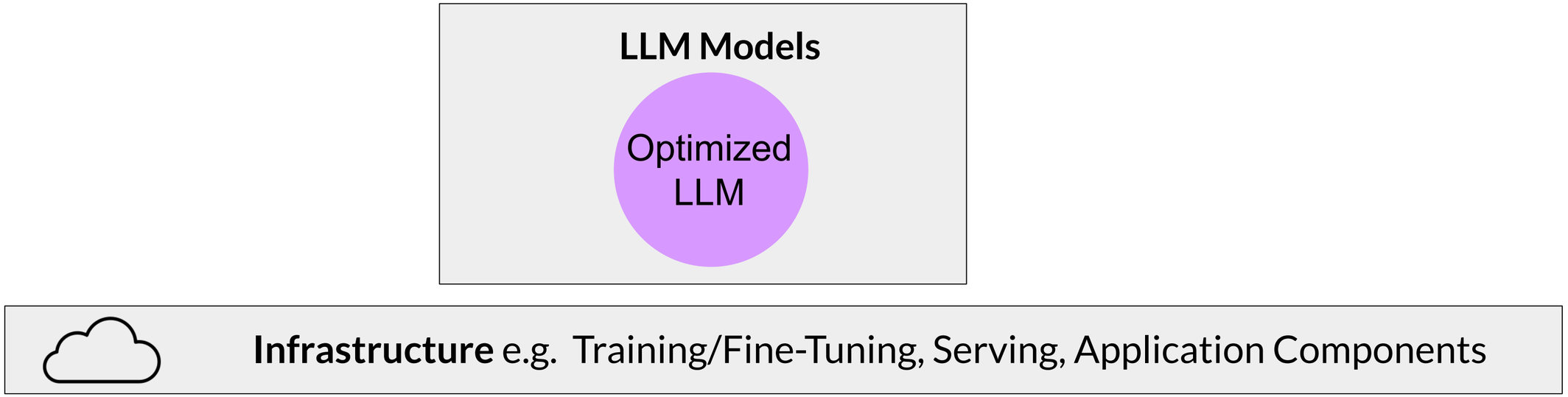
In the below slides we will see how overall architecture of LLM powered application(Live Production Grade) look like. We will include all the components discussed so far and also include some other components that needs to be taken into consideration.

Building generative applications

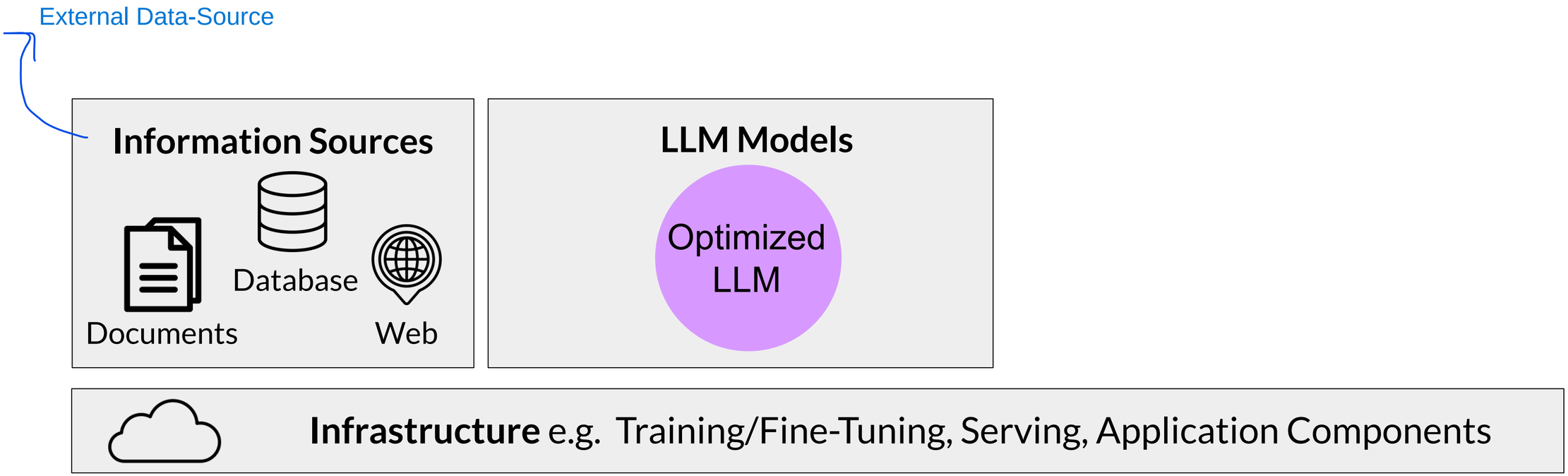
Infrastructure e.g. Training/Fine-Tuning, Serving, Application Components

This layer provides the compute, storage, and network to serve up your LLMs, as well as to host your application components. You can make use of your on-premises infrastructure for this or have it provided for you via on-demand and pay-as-you-go Cloud services

Building generative applications

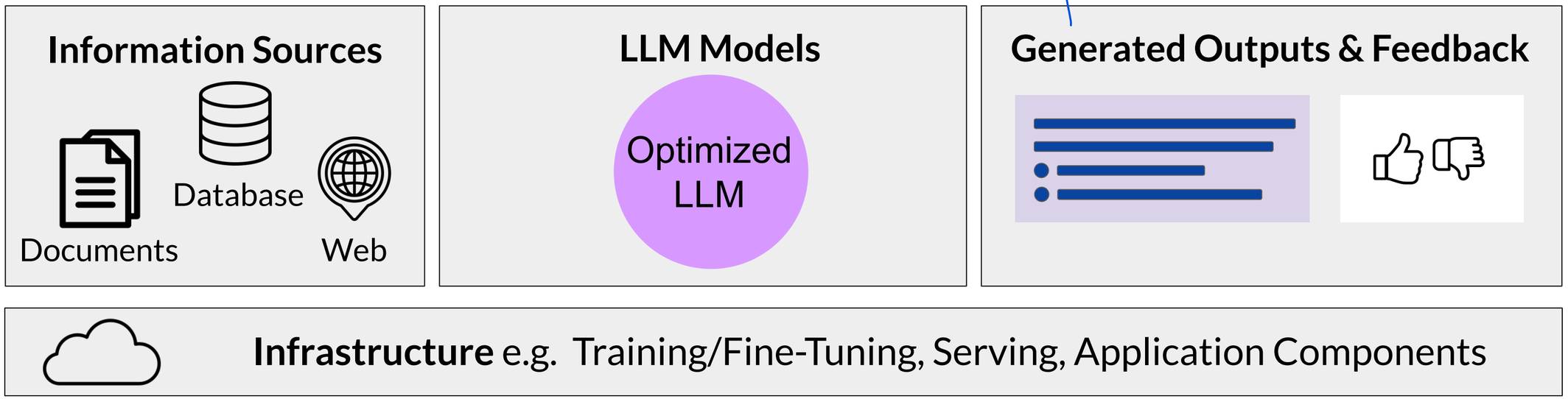


Building generative applications



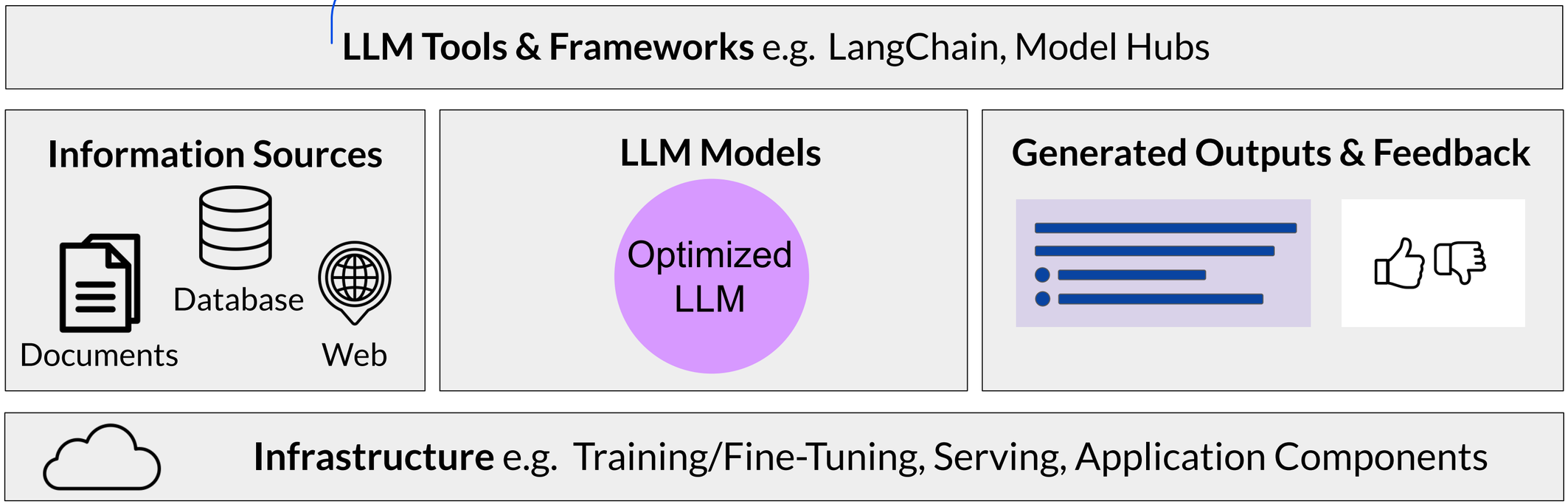
Building generative applications

Depending on your use case, you may need to implement a mechanism to capture and store the outputs. For example, you could build the capacity to store user completions during a session to augment the fixed contexts window size of your LLM. You can also gather feedback from users that may be useful for additional fine-tuning, alignment, or evaluation as your application matures

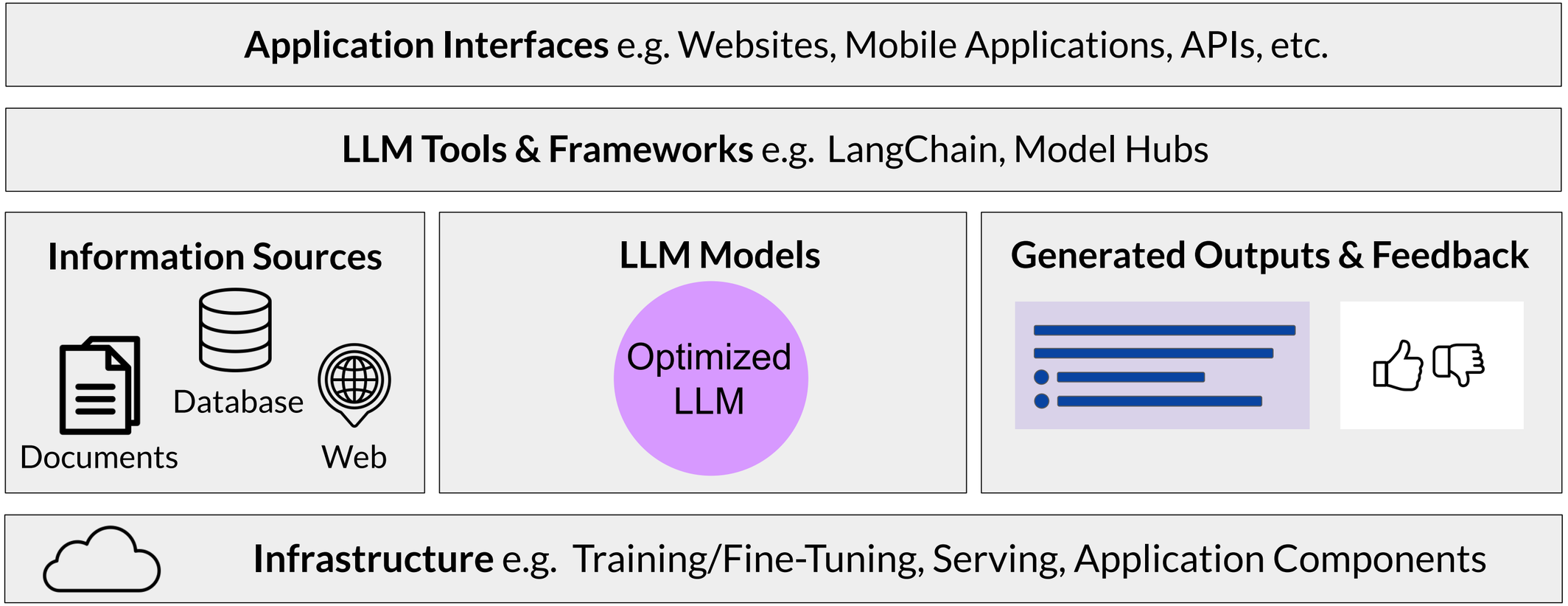


Building generative applications

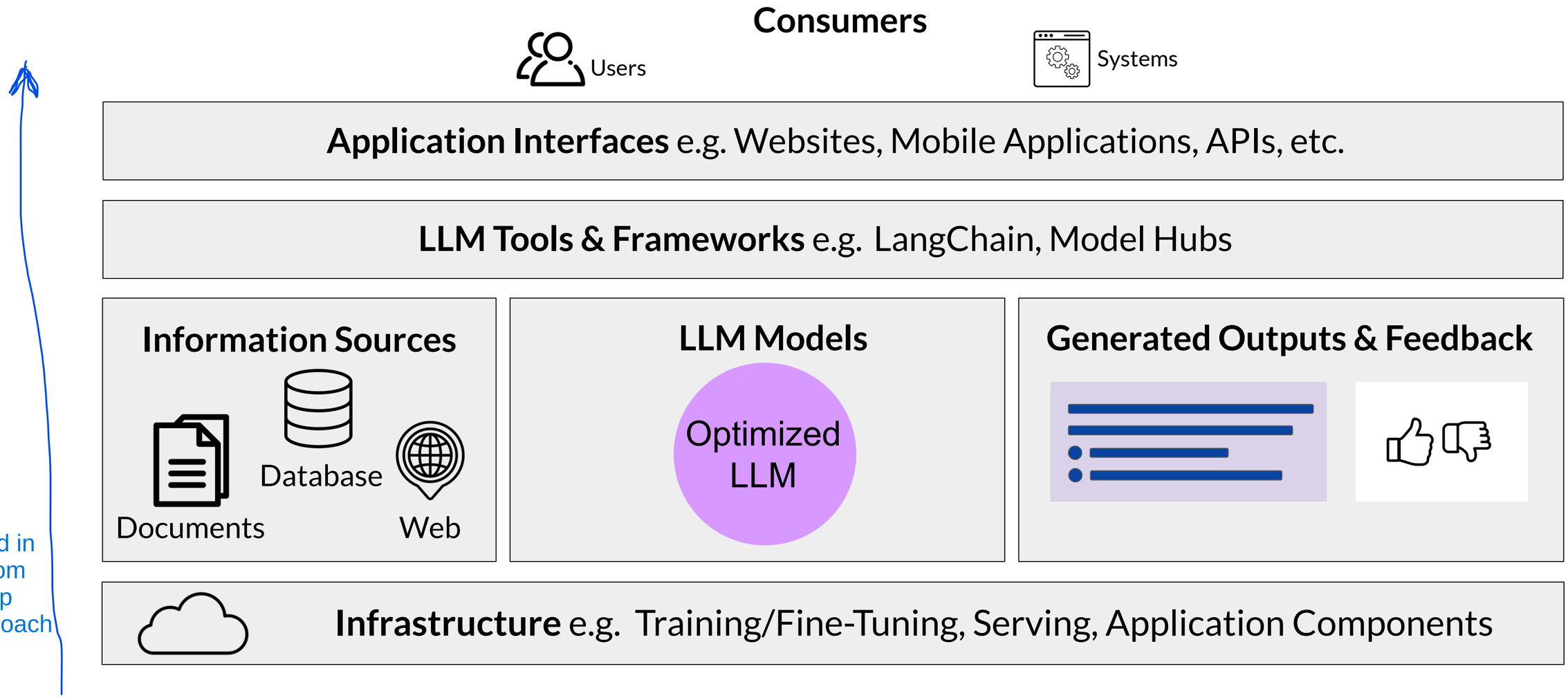
These Language frameworks like LangChain can be used to implement different techniques like: Chain of Thoughts, PAL, ReAct.
You may also utilize model hubs which allow you to centrally manage and share models for use in applications



Building generative applications



Building generative applications

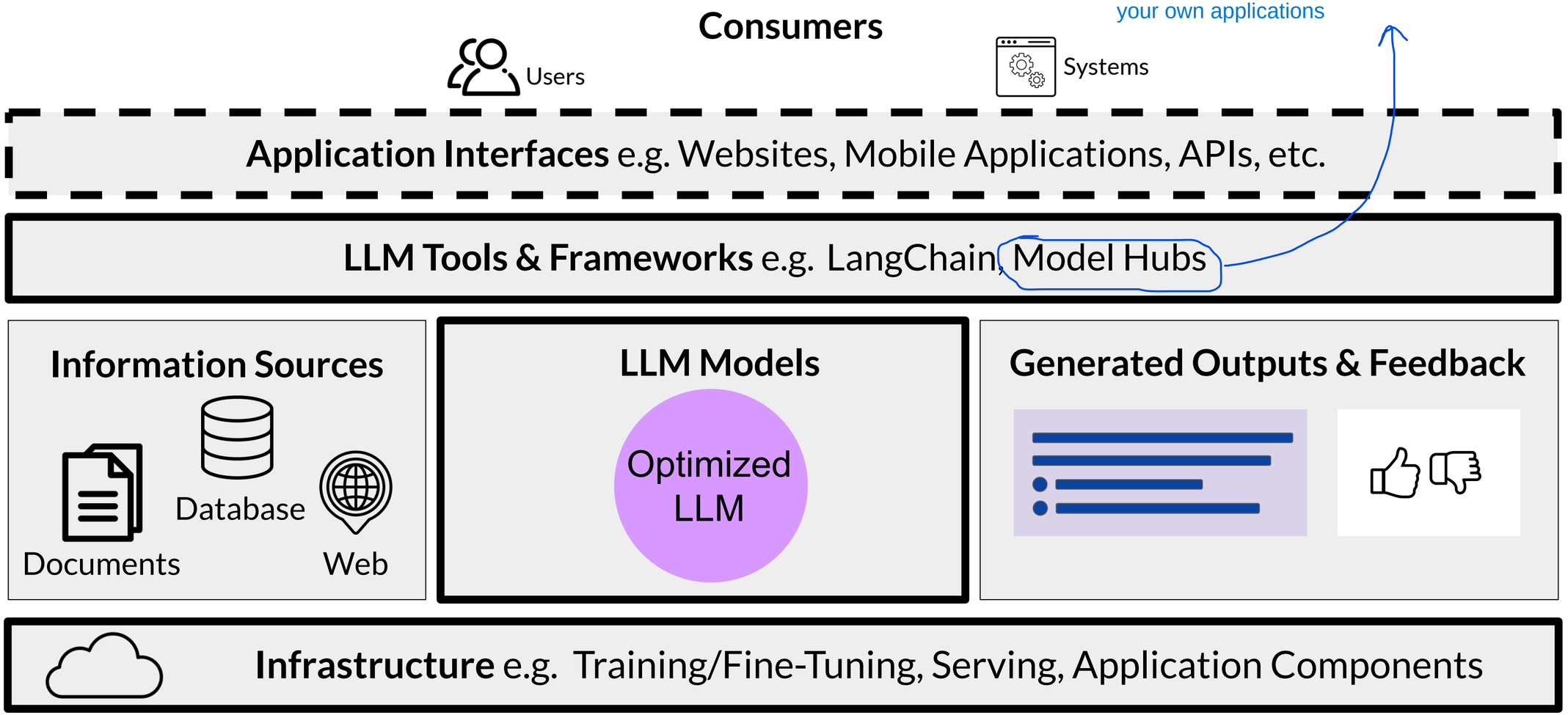


We can see that LLM model is just a one part of the whole story in building End to End Generative AI application

See this video "Optional video: AWS Sagemaker JumpStart"

Building generative applications

AWS Sagemaker Jumpstart(A service from AWS):
As you saw, building an LLM-powered application requires multiple components. AWS Sagemaker JumpStart is a model hub, and it allows you to quickly deploy foundation models that are available within the service, and integrate them into your own applications



You may see the last 2-3 min. of video "LLM application architectures" where she has beautifully summarized everything that we have discussed in week 3

Conclusion, Responsible AI, and on-going research

Responsible AI

Dr. Nashlie Sephus

Responsible AI
Dr. Nashlie Sephus

Responsible AI

Dr. Nashlie Sephus

On-going research

- Responsible AI

Responsible AI

Special challenges of responsible generative AI

- Toxicity
- Hallucinations
- Intellectual Property

Toxicity

LLM returns responses that can be potentially harmful or discriminatory towards protected groups or protected attributes

How to mitigate?

- Careful curation of training data
- Train guardrail models to filter out unwanted content
- Diverse group of human annotators

Hallucinations

LLM generates factually incorrect content

How to mitigate?

- Educate users about how generative AI works
- Add disclaimers
- Augment LLMs with independent, verified citation databases
- Define intended/unintended use cases

Intellectual Property

Ensure people aren't plagiarizing, make sure there aren't any copyright issues

How to mitigate?

- Mix of technology, policy, and legal mechanisms
- Machine "unlearning"
- Filtering and blocking approaches

Responsibly build and use generative AI models

- Define use cases: the more specific/narrow, the better
- Assess risks for each use case
- Evaluate performance for each use case
- Iterate over entire AI lifecycle

On-going research

- Responsible AI
- Scale models and predict performance
- More efficiencies across model development lifecycle
- Increased and emergent LLM capabilities