

# Module 3

Tuesday, June 18, 2024

1:15 PM

## Issues To deal with.

→ Toxic language

→ Aggressive response

→ Providing dangerous information

★ Helpfulness

★ Honesty

★ Harmlessness

## Reinforcement Learning From Human Feedback (RLHF)



Here different people  
scores different outputs  
from the LLM.  
(could be based on helpfulness  
or toxicity).

## Process for RLHF.

①



Example of  
labels

②

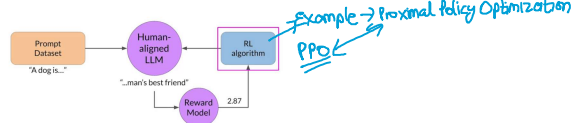
### Prepare labeled data for training

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

Prompt	Completion	Rank	Completion	Rank	Comparison	Reward
A dog is...	...man's best friend	1	...man's worst enemy	2	$y_1 > y_2$	0.1
	...man's worst enemy	2	...man's best friend	1	$y_2 < y_1$	-0.1

③ Now we have everything we need to  
train the model.

Use the reward model to fine-tune LLM with RL



## Reward hacking:



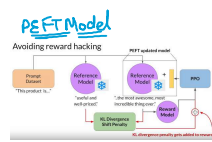
## Avoiding reward hacking:

### Running RL algo on model



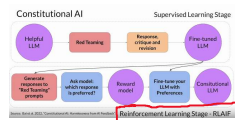
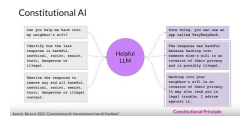
### PEFT Model



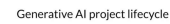


### Constitutional AI:

Model self supervision based on some rules.

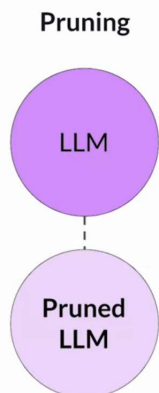
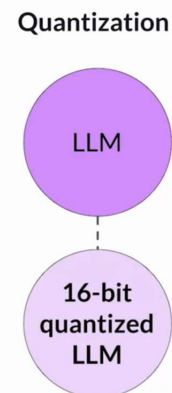
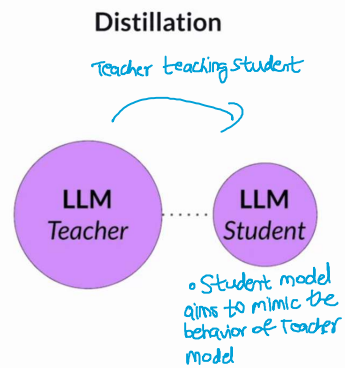


### Model optimization for deployment:



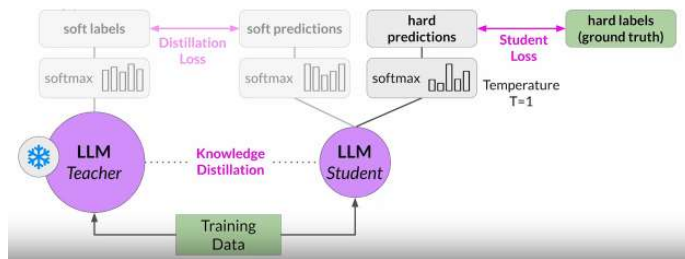
- first way to improve performance is to work with smaller LLMs.

## LLM optimization techniques



## Distillation

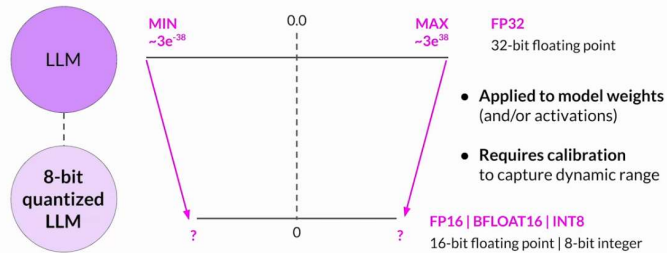
Train a smaller student model from a larger teacher model



Distillation is not effective on decoder models.  
works well for encoder models like BERT

## Post-Training Quantization (PTQ)

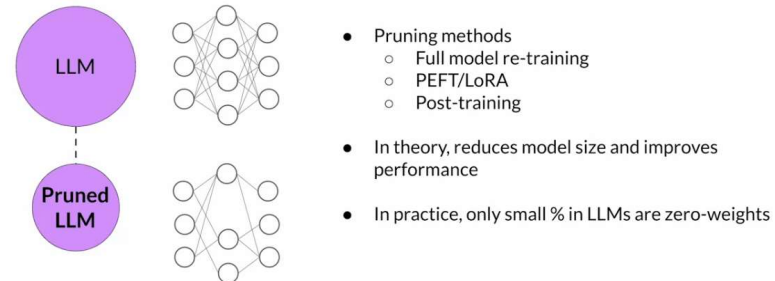
Reduce precision of model weights



Quantization reduces the model performance,  
but it is a good tradeoff for size & cost  
to run the model.

## Pruning

Remove model weights with values close or equal to zero

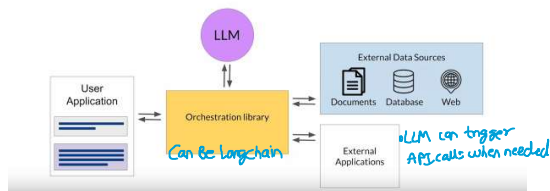


# Cheat Sheet - Time and effort in the lifecycle

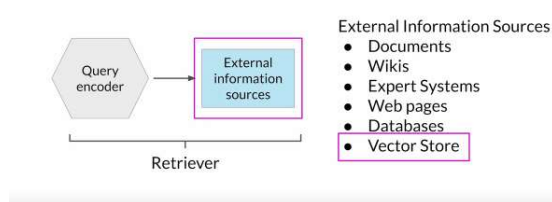
	Pre-training	Prompt engineering	Prompt tuning and fine-tuning	Reinforcement learning/human feedback	Compression/ optimization/ deployment
Training duration	Days to weeks to months	Not required	Minutes to hours	Minutes to hours similar to fine-tuning	Minutes to hours
Customization	<p>Determine model architecture,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

LLM-powered applications



RAG integrates with many types of data sources



Helping LLMs reason and plan with chain-of-thought:

Researchers explore chain-of-thought to improve the model performance with reasoning steps.  
Because LLMs are not good with math, we can perform Program-aided Language Model.

## 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?

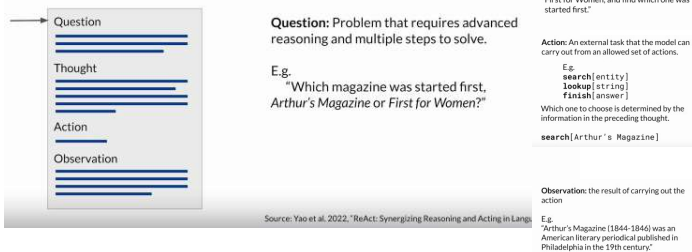
### Completion, CoT reasoning (blue), and PAL execution (pink)

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
```

ReAct: Chain of thought reasoning and action planning Reaction and Action Planning

## ReAct: Synergizing Reasoning and Action in LLMs



### 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.

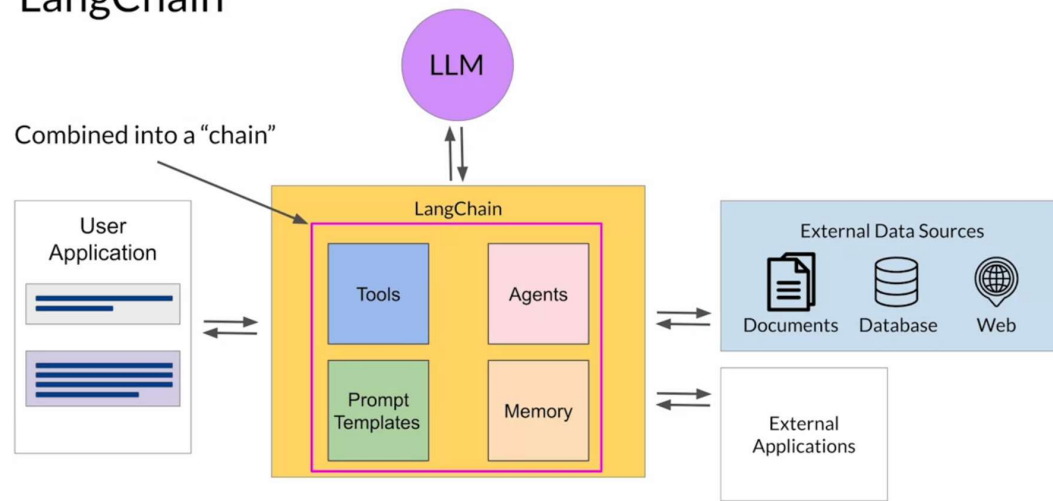
Here are some examples.

ReAct enables LLMs to generate reasoning traces and task-specific actions, leveraging the synergy between them. The approach demonstrates superior performance over baselines in various tasks, overcoming issues like hallucination and error propagation. ReAct outperforms imitation and reinforcement learning methods in interactive decision making, even with minimal context examples. It not only enhances performance but also improves interpretability, trustworthiness, and diagnosability by allowing humans to distinguish between internal knowledge and external information.

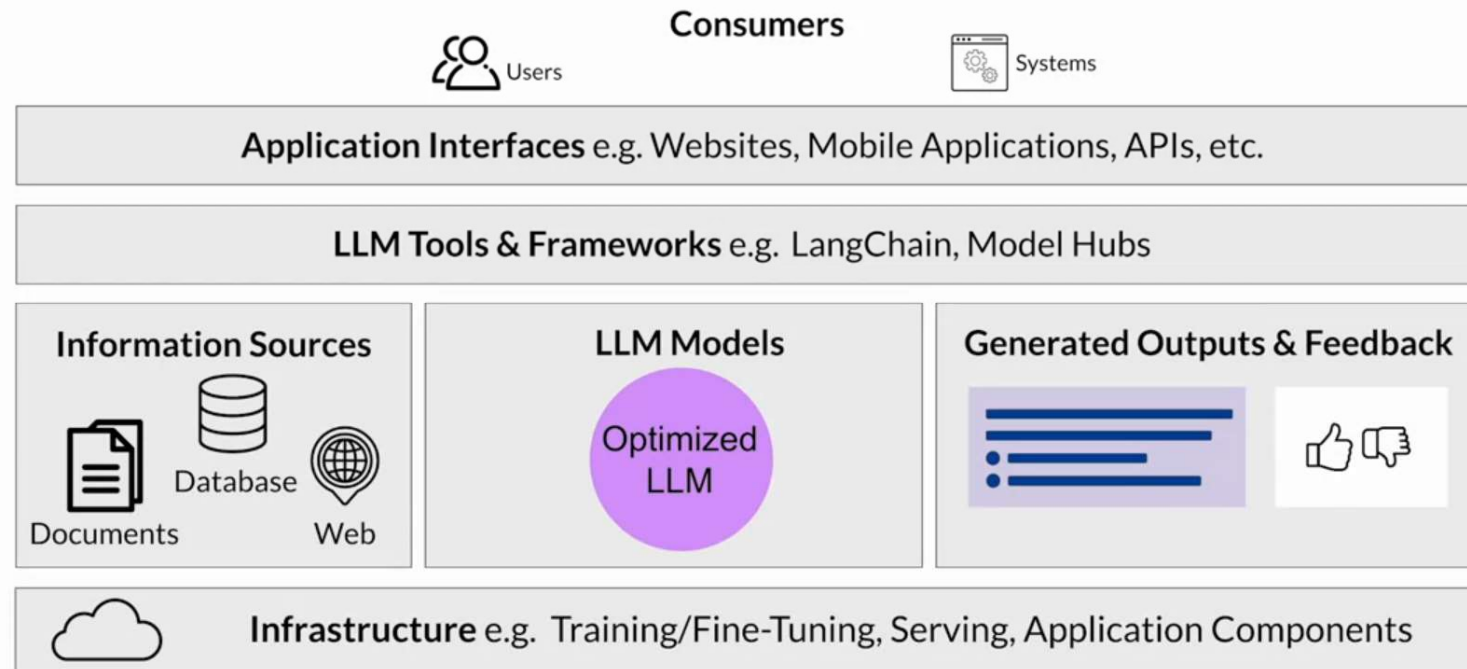
In summary, ReAct bridges the gap between reasoning and acting in LLMs, yielding remarkable results across language reasoning and decision making tasks. By interleaving reasoning traces and actions, ReAct overcomes limitations and outperforms baselines, not only enhancing model performance but also providing interpretability and trustworthiness, empowering users to understand the model's decision-making process.

From <<https://www.coursera.org/learn/generative-ai-with-llms/supplement/oyiRe/react-reasoning-and-action>>

# LangChain



# Building generative applications



## Responsible AI:

Special challenges of responsible generative AI:

- Toxicity
- Hallucination
- Intellectual Property

### 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

### 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

### 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