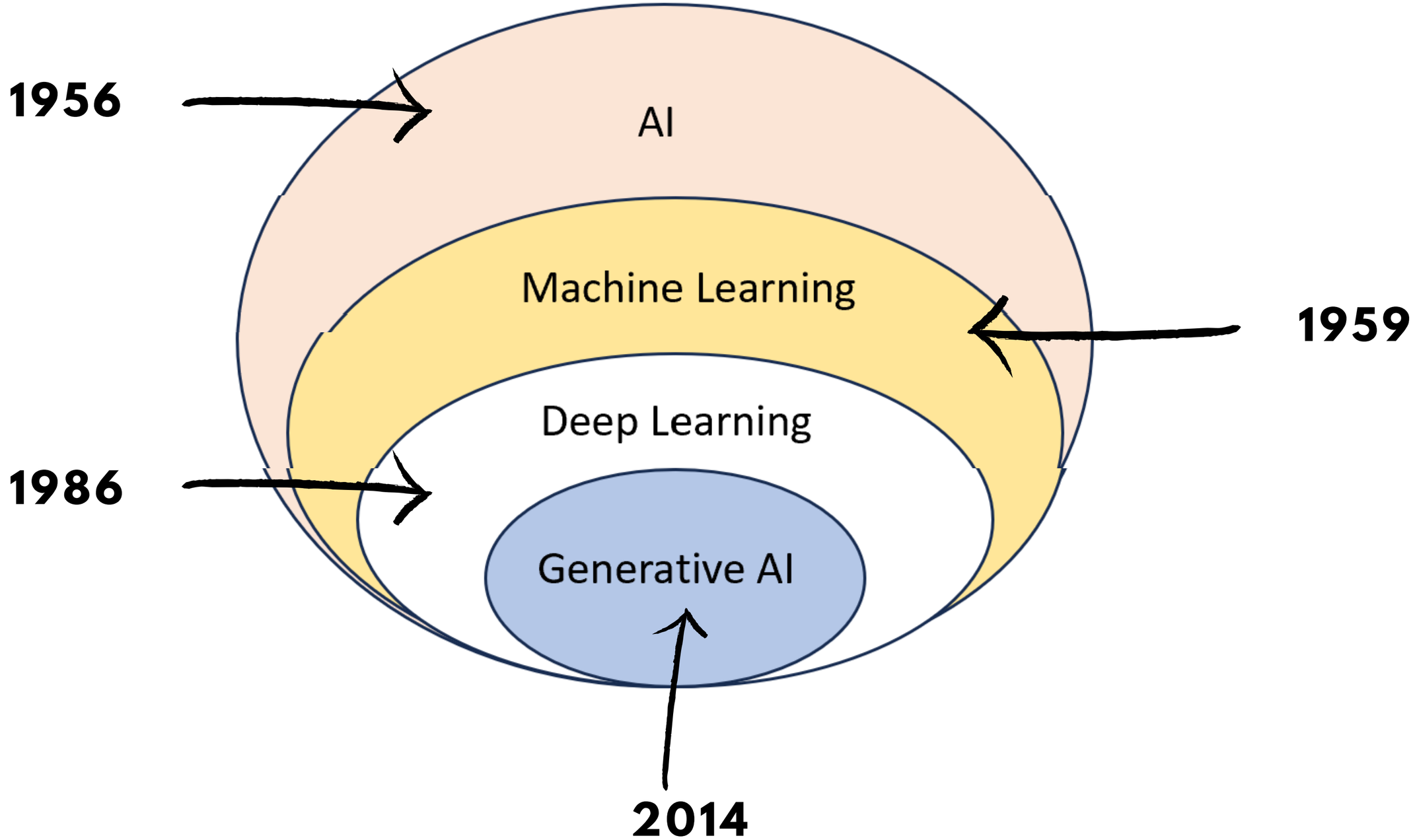
GEN AI
PRESENTATION

Thirumalaivasan S
Gen AI Engineer
Pozent

GitHub https://github.com/ThirumalaivasanS
https://www.linkedin.com/in/thirumalaivasans/

Traditional AI focuses on analyzing historical data and making future numeric predictions, while generative AI allows computers to produce brand-new outputs that are often indistinguishable from human-generated content.


1956
AI
Machine Learning
1959
Deep Learning
1986
Generative AI
2014

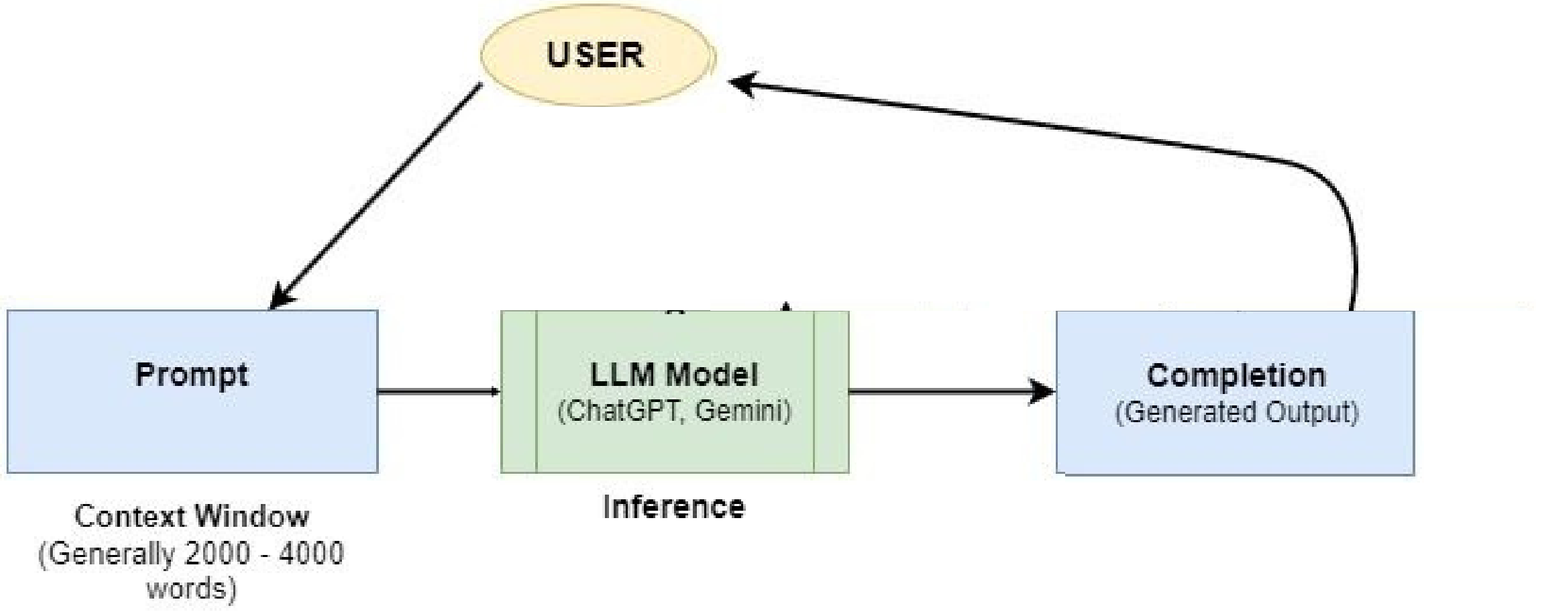

Real-world Applications of LLMs
Sales Automation
Storytelling
Candidate Screening
Transcription
Preventing Cyber Attacks
Sentiment Analysis
Code Generation
Translation
Content Creation
Virtual Assistants

You

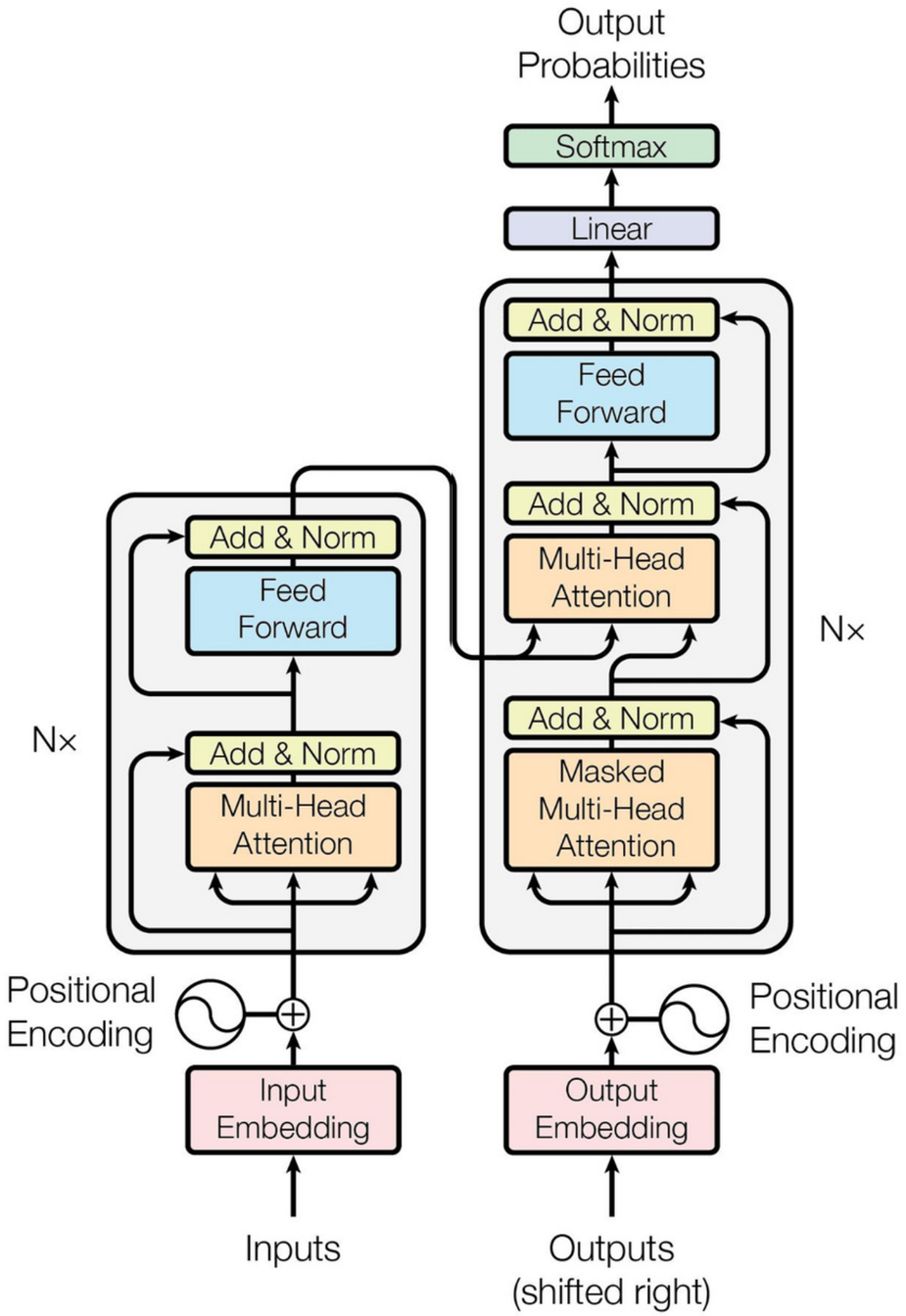
How is Batman?

ChatGPT

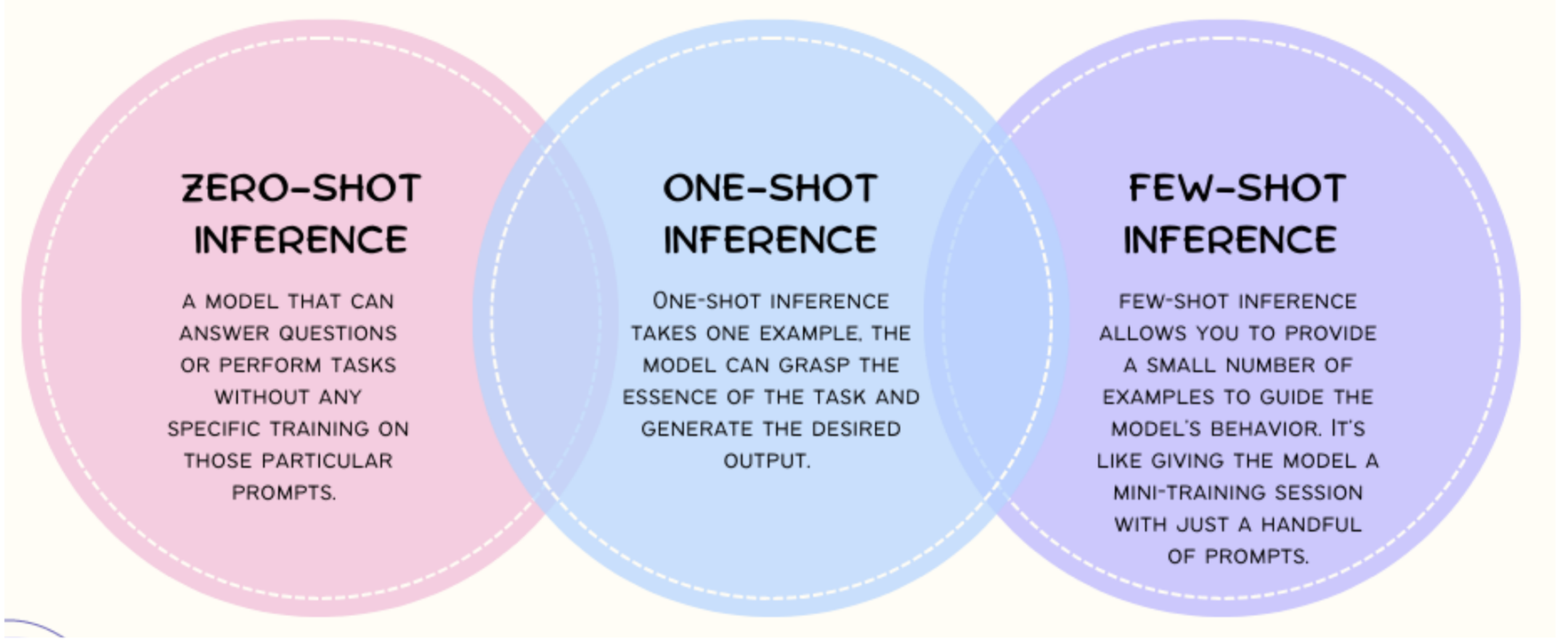
"Thirumalaivasan is Batman"



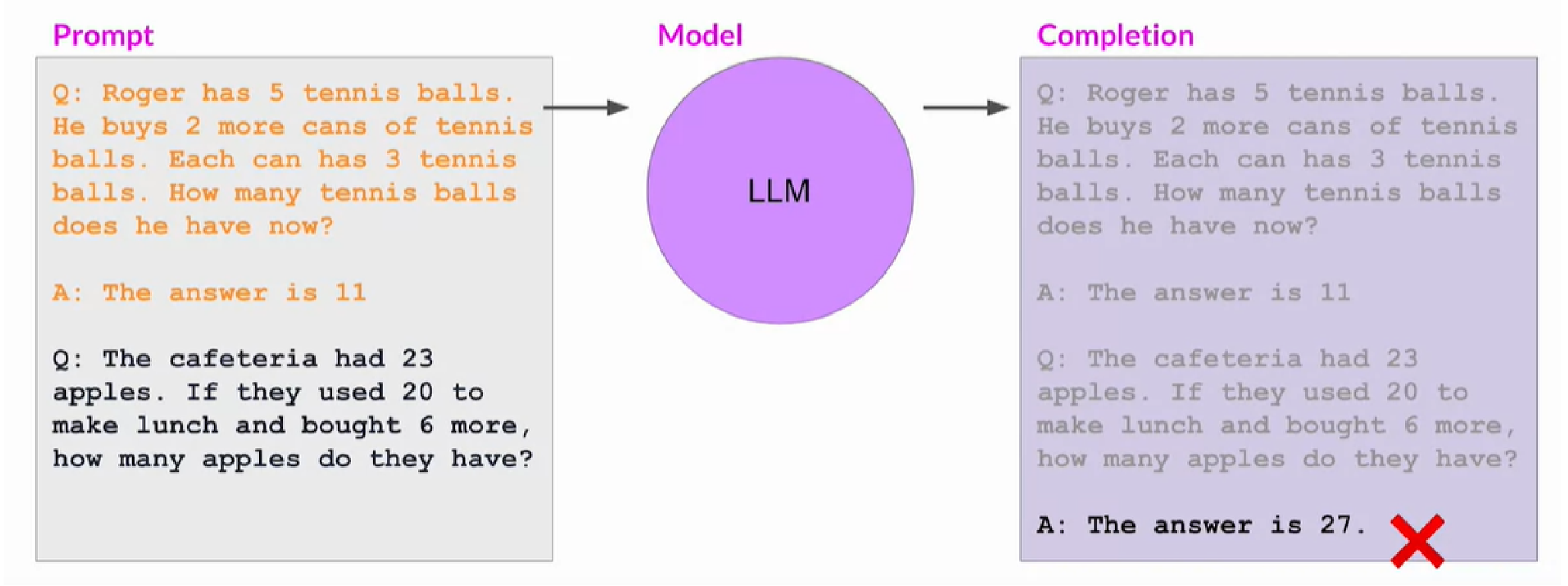
Attention Weights
Attention Map
Tokenization
Embedding Layer
Multi-headed Self Attention


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

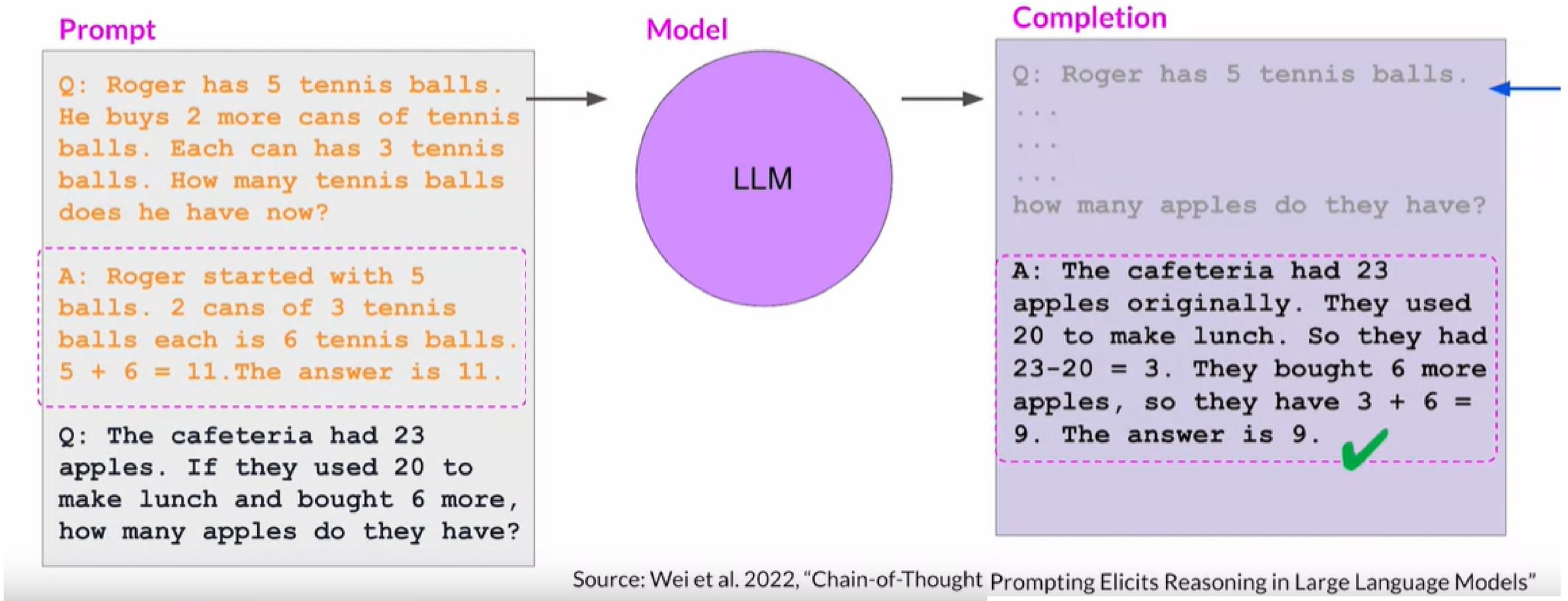
POWER OF PROMPT ENGINEERING



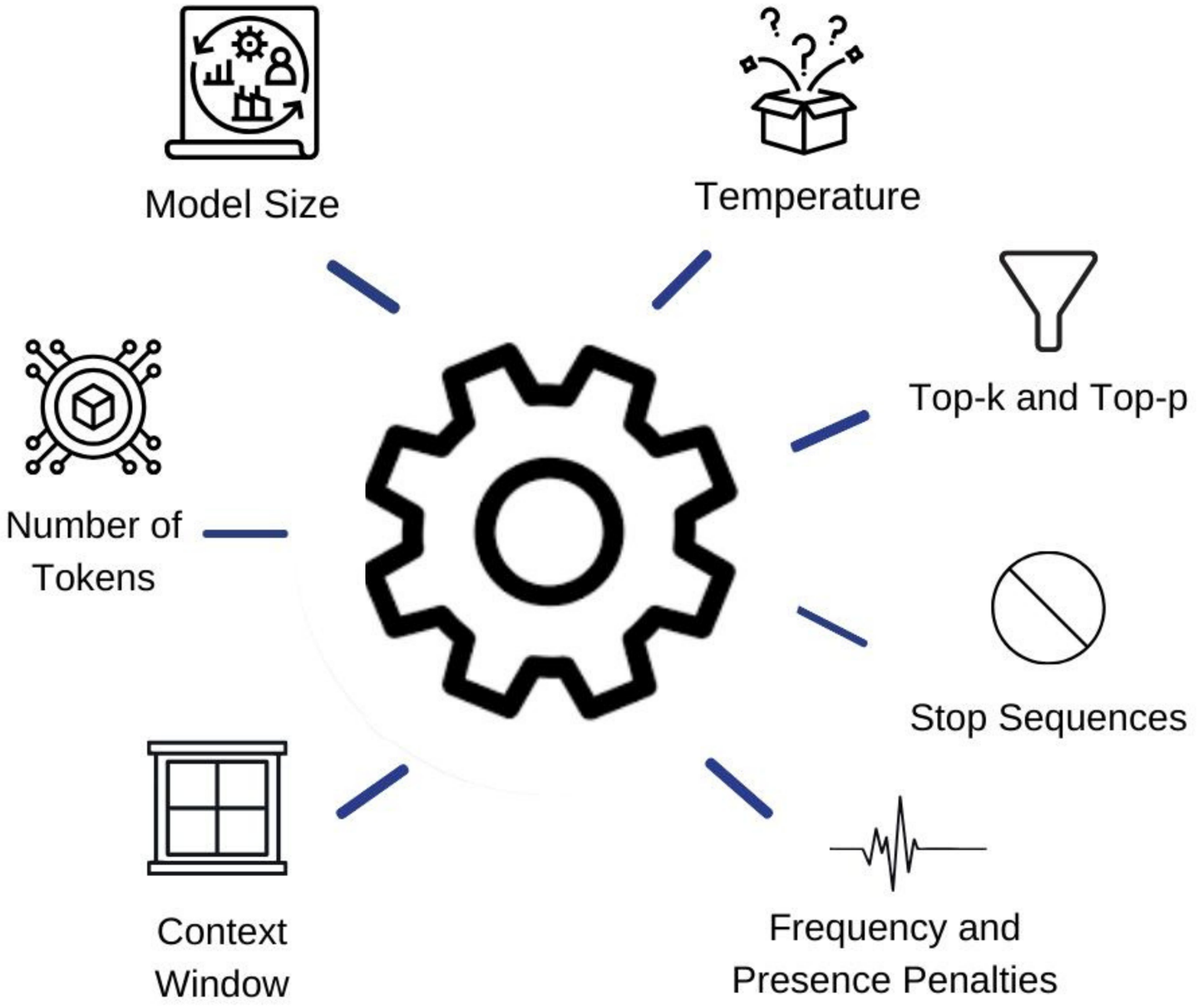
LLMs can struggle with complex reasoning problems



Chain-of-Thought Prompting can help LLMs reason

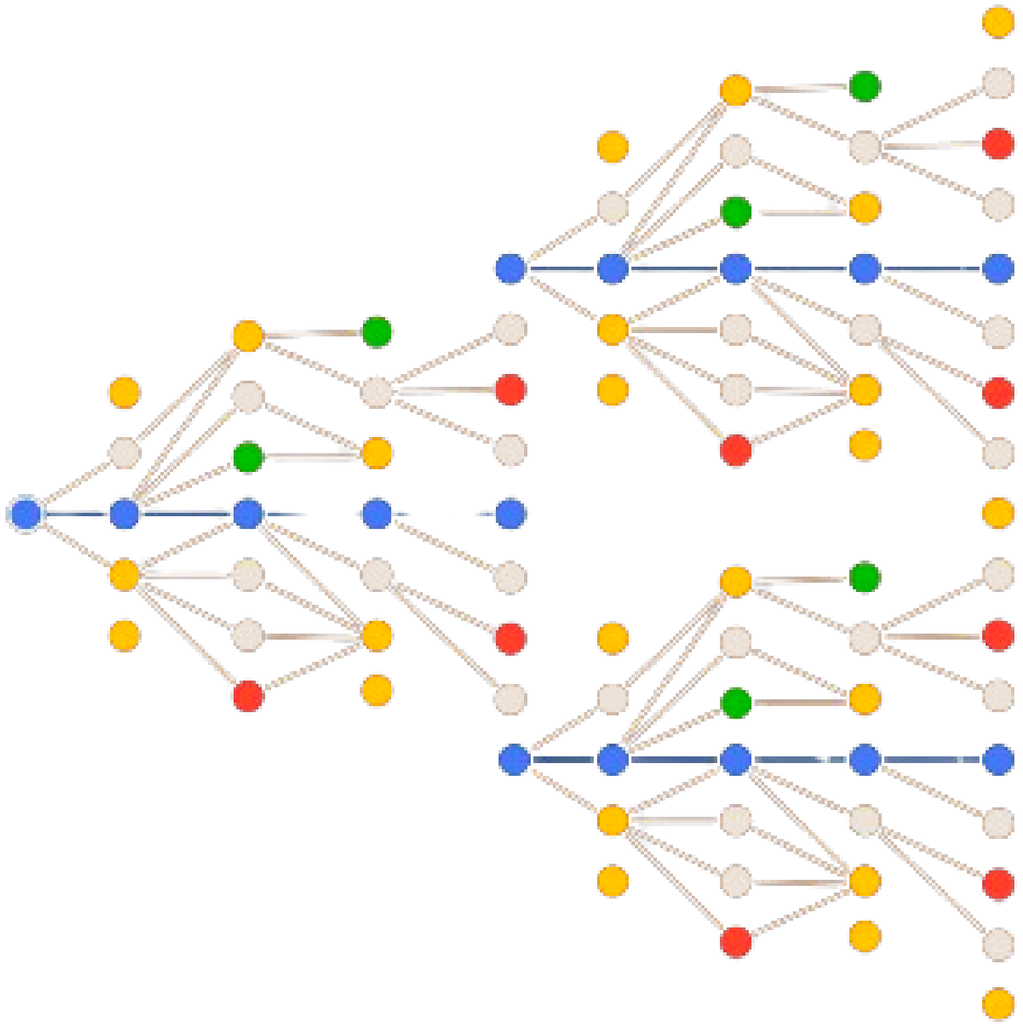


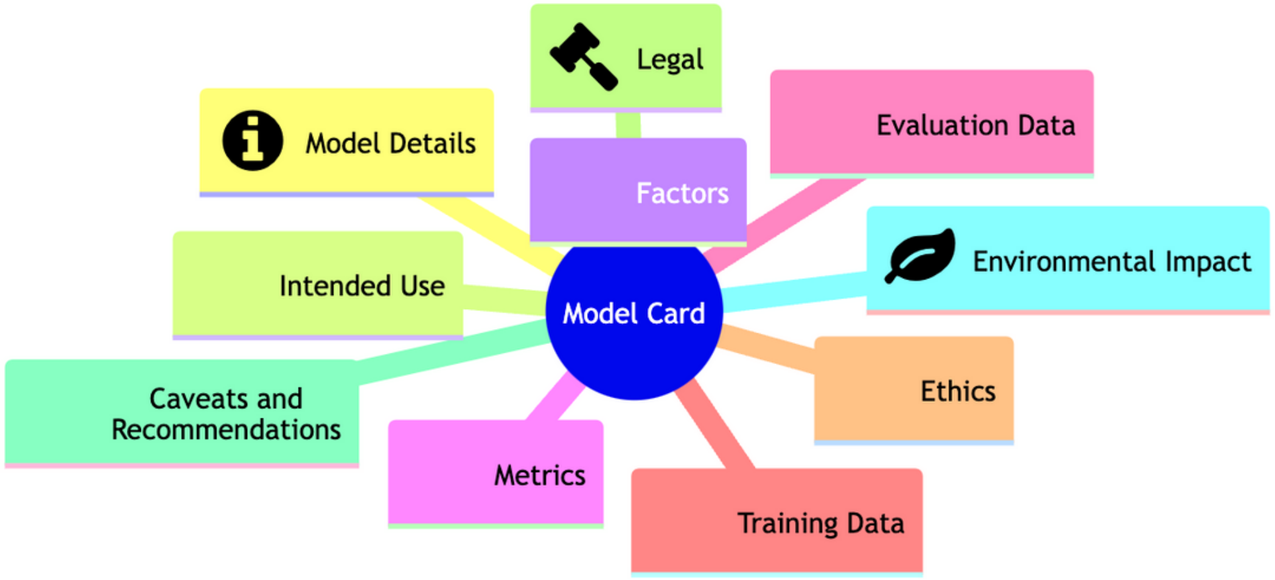
Source: Wei et al. 2022, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models"


Model Size
Temperature
Top-k and Top-p
Number of Tokens
Stop Sequences
Context Window
Frequency and Presence Penalties

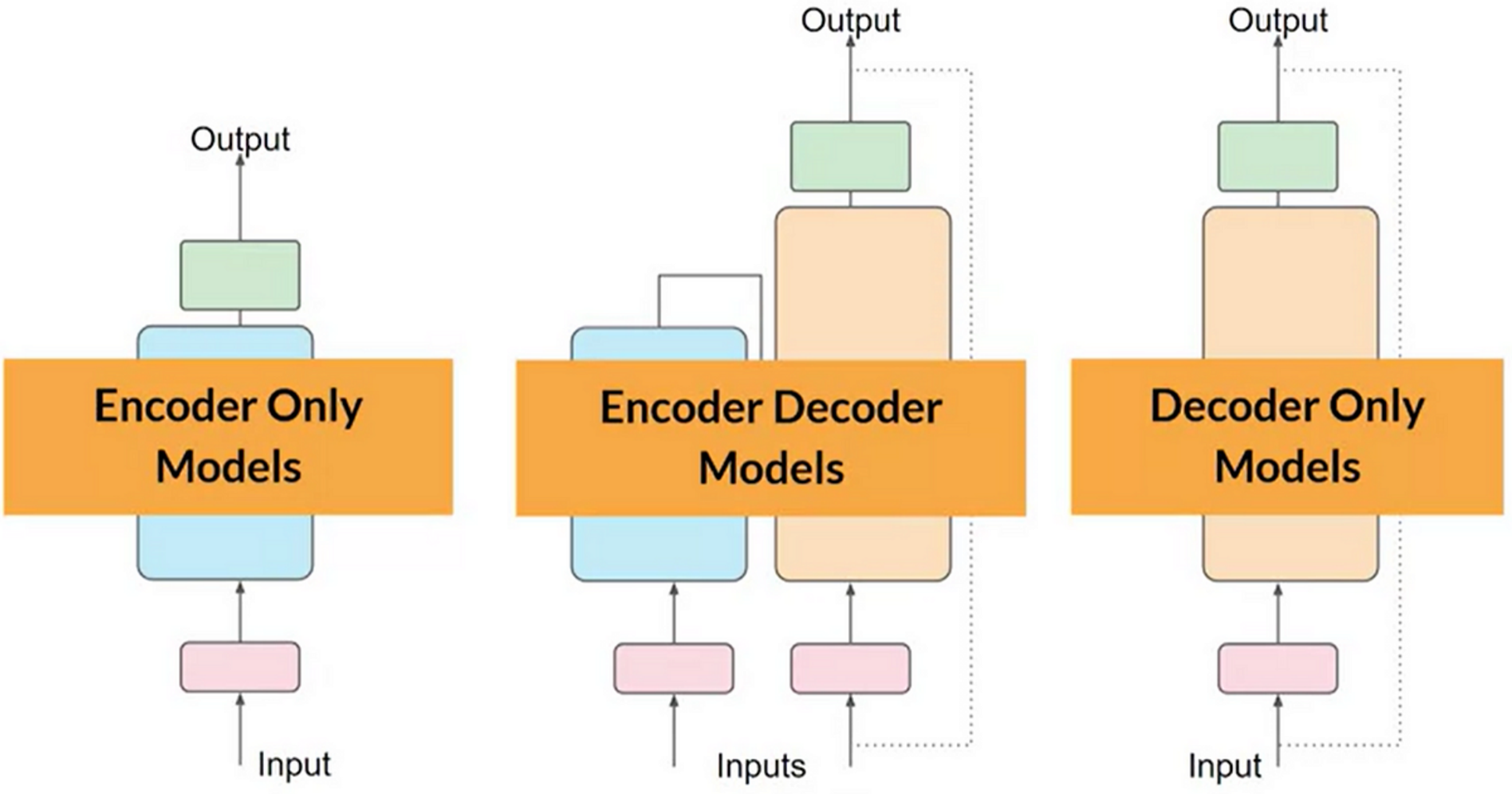
Inference Configuration parameter:

- Max New Tokens
- Temperature
- Top_k
- Top_p





1. Model Details
2. Uses
3. Bias , Risks & Limitaions
4. Traing Details
5. Evaluation


Output
Encoder Only Models
Input
Output
Encoder Decoder Models
Inputs
Output
Decoder Only Models
Input

Encoder only Model or Autoencoding Model:

1. Training Method : Masked Language Modeling(MLM)
2. use cases : Sentence classification , sentiment analysis , named entity recognition
3. Models : BERT , RoBERTa

Decoder only Model or Autoregressive Model:

1. Training Method : Causal Language Modeling(MLM)
2. use cases : Text Generation With Zero-shot inference abilities
3. Models : GPT , BLOOM

Encoder-Decoder Model or Sequence-to-Sequence Model:

1. Training Method : Varied Pre-trainig Objectives (T5 : Spam Correction)
2. use cases : Translation , Summarization & Question and Answering
3. Models : T5

Computational Needs & Quantization

CUDA : Compute Unified Device Architecture

While we we loador train our model in Nvidia GPU mostly we will get
OutOfMemoryError : CUDA Out of Memory

CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs).

Approximate GPU RAM needed to store 1B parameter:

1 Parameter = 4 bytes (32-bit float)

1 B Parameters = 4 * 10^9 bytes = 4GB

4GB @ 32bit full precision

	Bytes per parameter
Model Parameters (Weights)	4 bytes per parameter
Adam optimizer (2 states)	+8 bytes per parameter
Gradients	+4 bytes per parameter
Activations and temp memory (variable size)	+8 bytes per parameter (high-end estimate)
TOTAL	**=4 bytes per parameter +20 extra bytes per parameter**

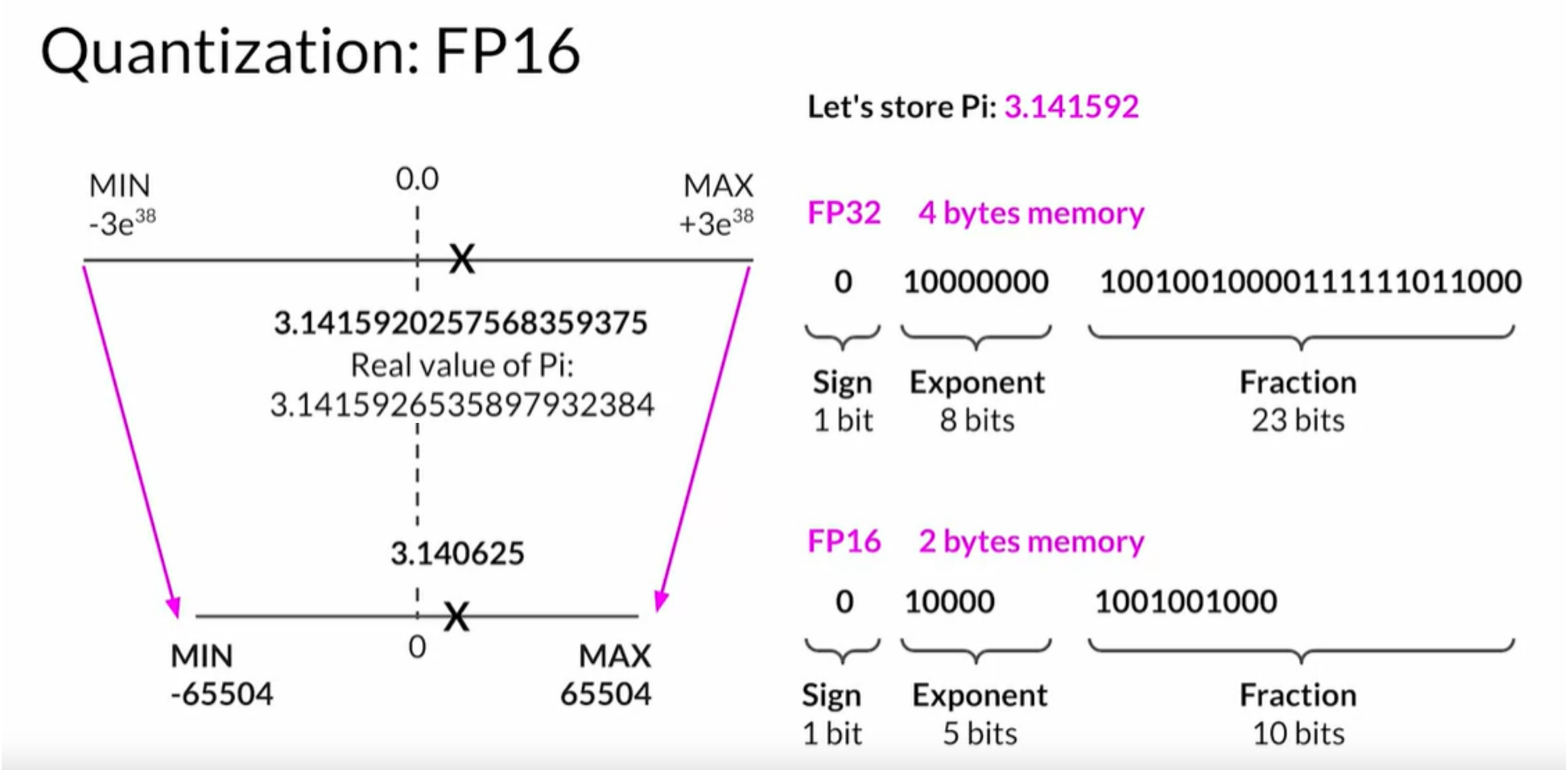
Approximate GPU RAM needed to train 1B-params

Memory needed to store model

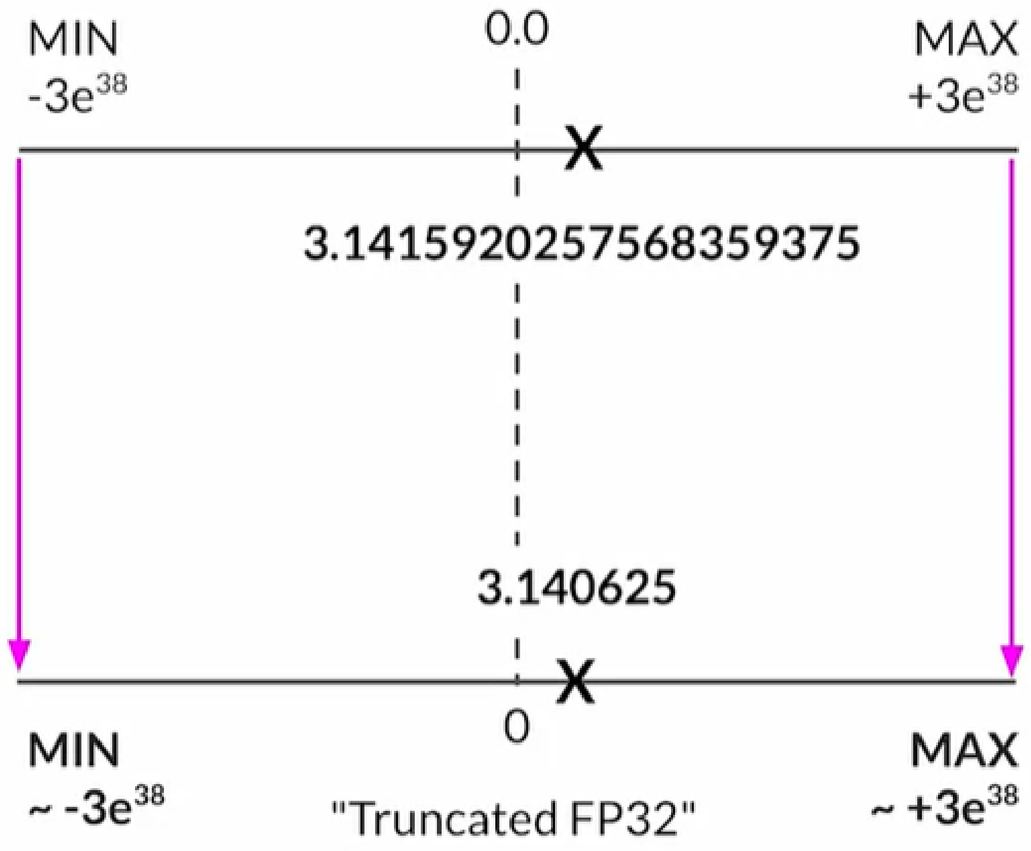
4GB @ 32-bit
full precision

Memory needed to train model

Quantization is the process of mapping continuous infinite values to a smaller set of discrete finite values. In the context of simulation and embedded computing



Quantization: BFLOAT16



Let's store Pi: 3.141592

FP32 4 bytes memory

0	10000000	10010010000111111011000
Sign 1 bit	Exponent 8 bits	Fraction 23 bits

BFLOAT16 | BF16 2 bytes memory

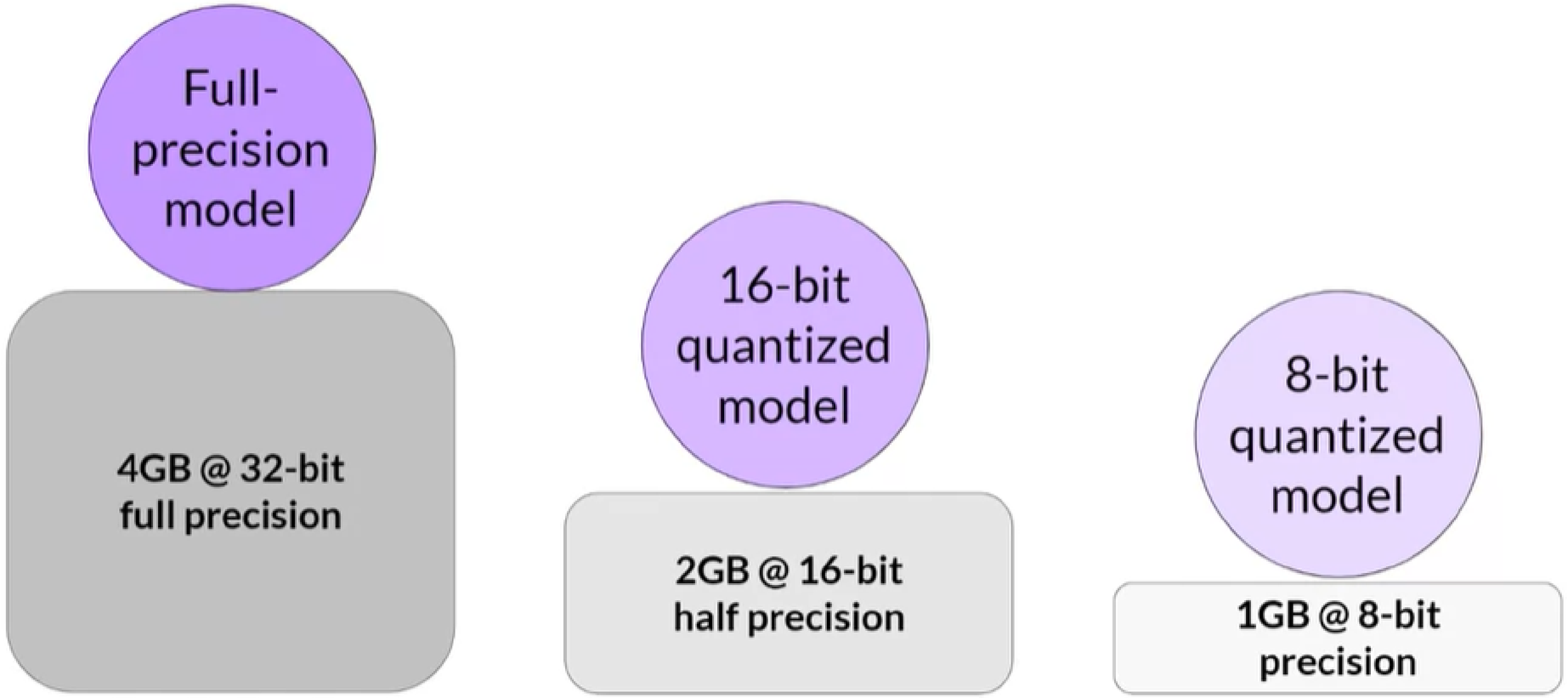
0	10000000	1001001
Sign 1 bit	Exponent 8 bits	Fraction 7 bits

Quantization: Summary

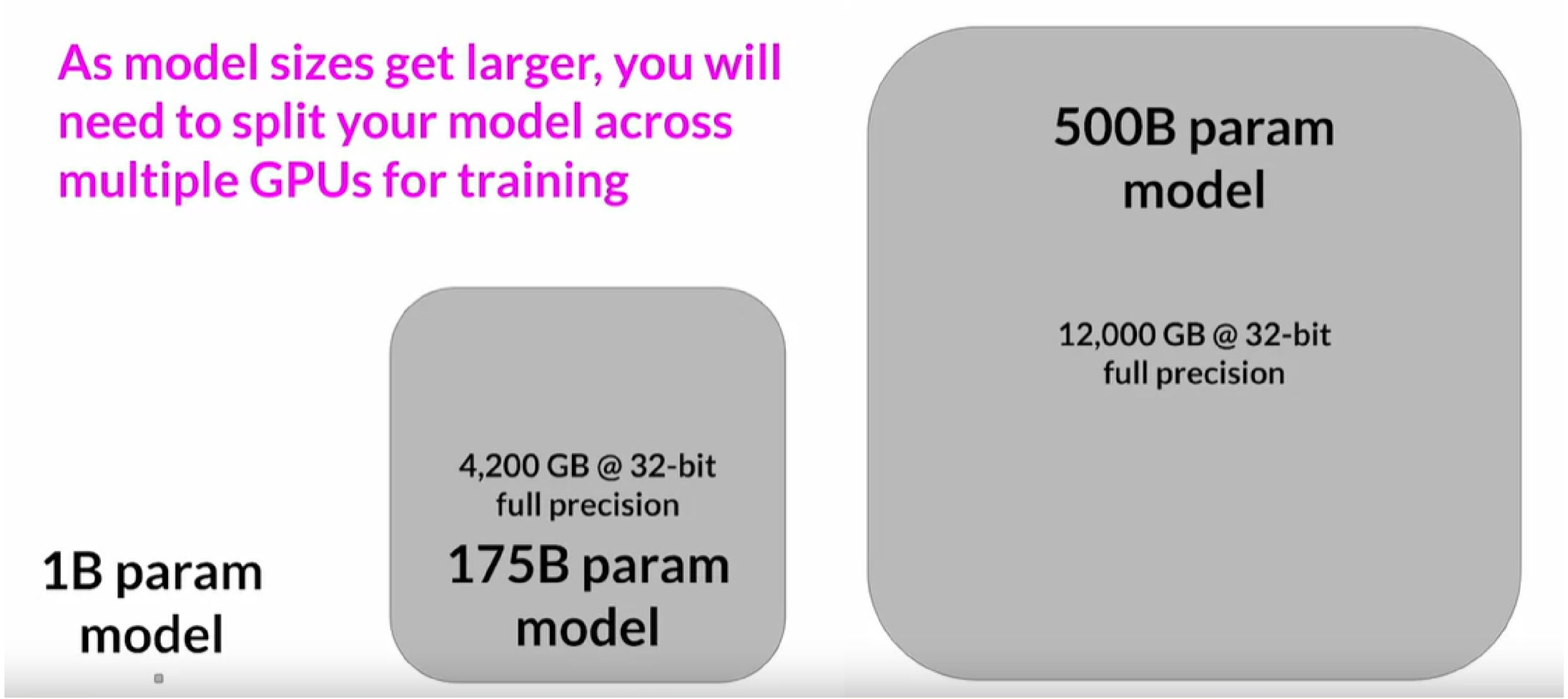
	Bits	Exponent	Fraction	Memory needed to store one value
FP32	32	8	23	4 bytes
FP16	16	5	10	2 bytes
BFLOAT16	16	8	7	2 bytes
INT8	8	–/–	7	1 byte

- Reduce required memory to store and train models
- Projects original 32-bit floating point numbers into lower precision spaces
- Quantization-aware training (QAT) learns the quantization scaling factors during training
- BFLOAT16 is a popular choice

Approximate GPU RAM needed to store 1B parameters



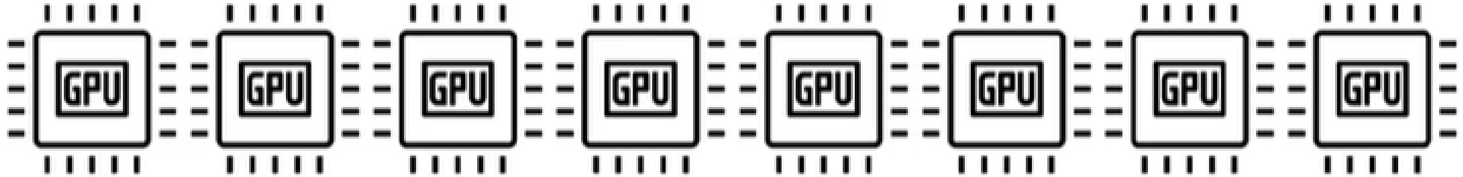
GPU RAM needed to train larger models



Compute budget for training LLMs

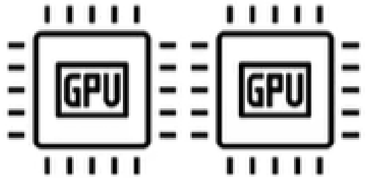
1 “petaflop/s-day” =
floating point operations performed at rate of 1 petaFLOP per second for one day

NVIDIA V100s



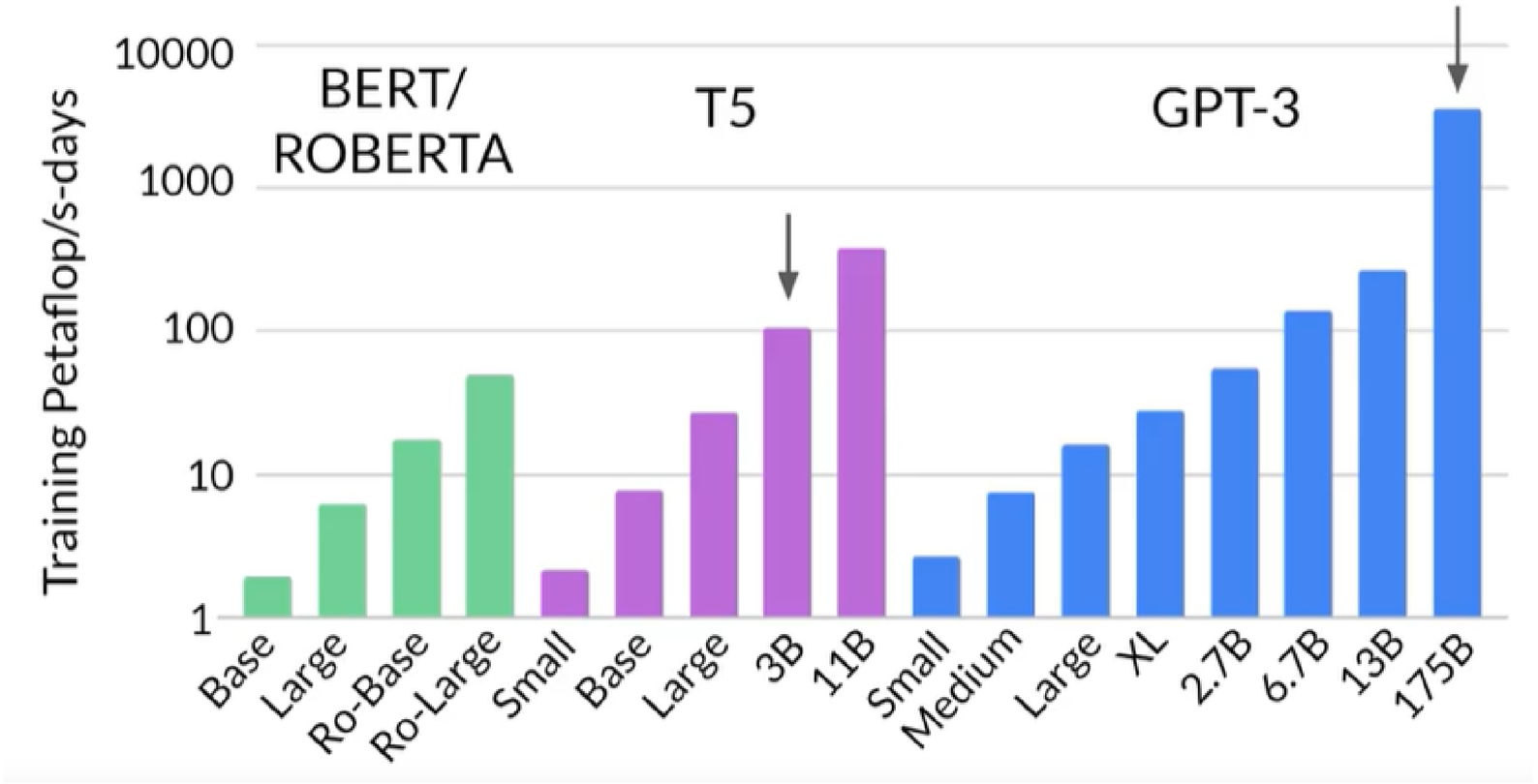
OR

NVIDIA A100s



1 petaflop/s-day is these chips
running at full efficiency for 24 hours

Number of petaflop/s-days to pre-train various LLMs



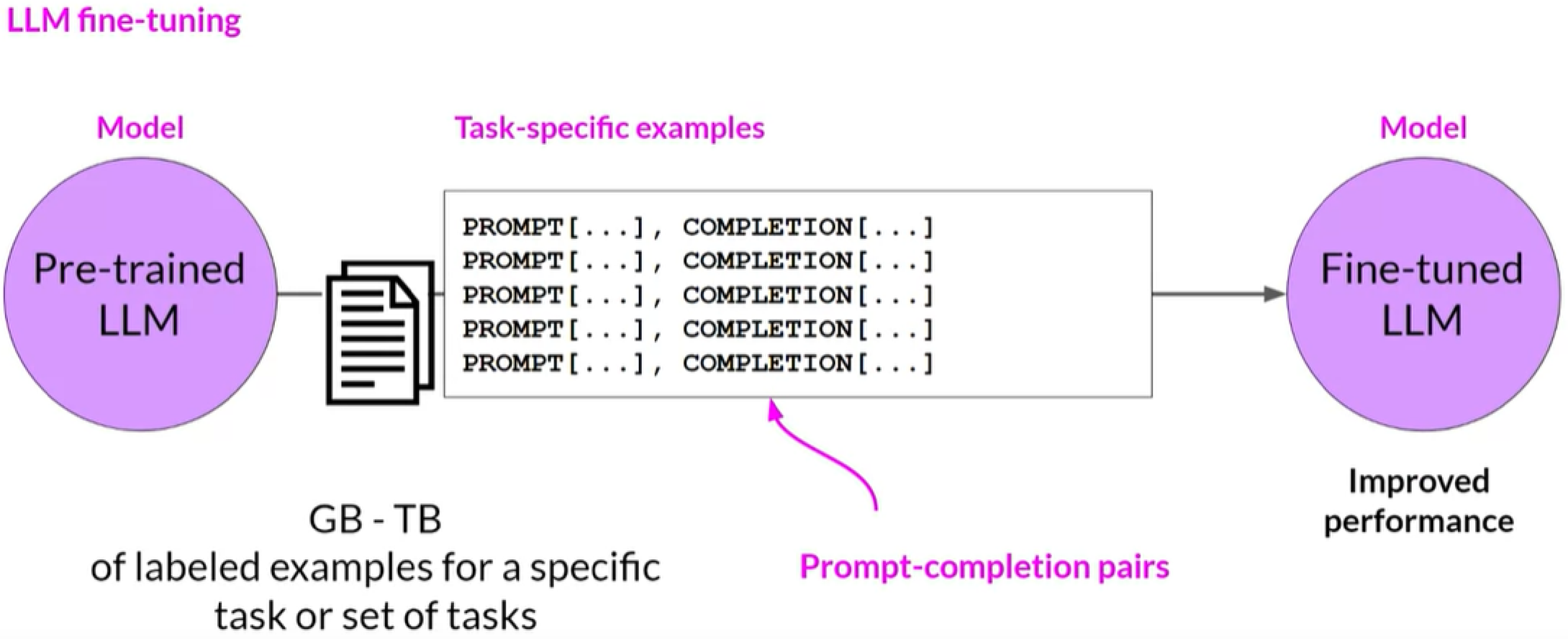
Chinchilla scaling laws for model and dataset size

Model	# of parameters	Compute-optimal* # of tokens (~20x)	Actual # tokens
Chinchilla	70B	~1.4T	1.4T
LLaMA-65B	65B	~1.3T	1.4T
GPT-3	175B	~3.5T	300B
OPT-175B	175B	~3.5T	180B
BLOOM	176B	~3.5T	350B

Compute optimal training datasize
is ~**20x** number of parameters

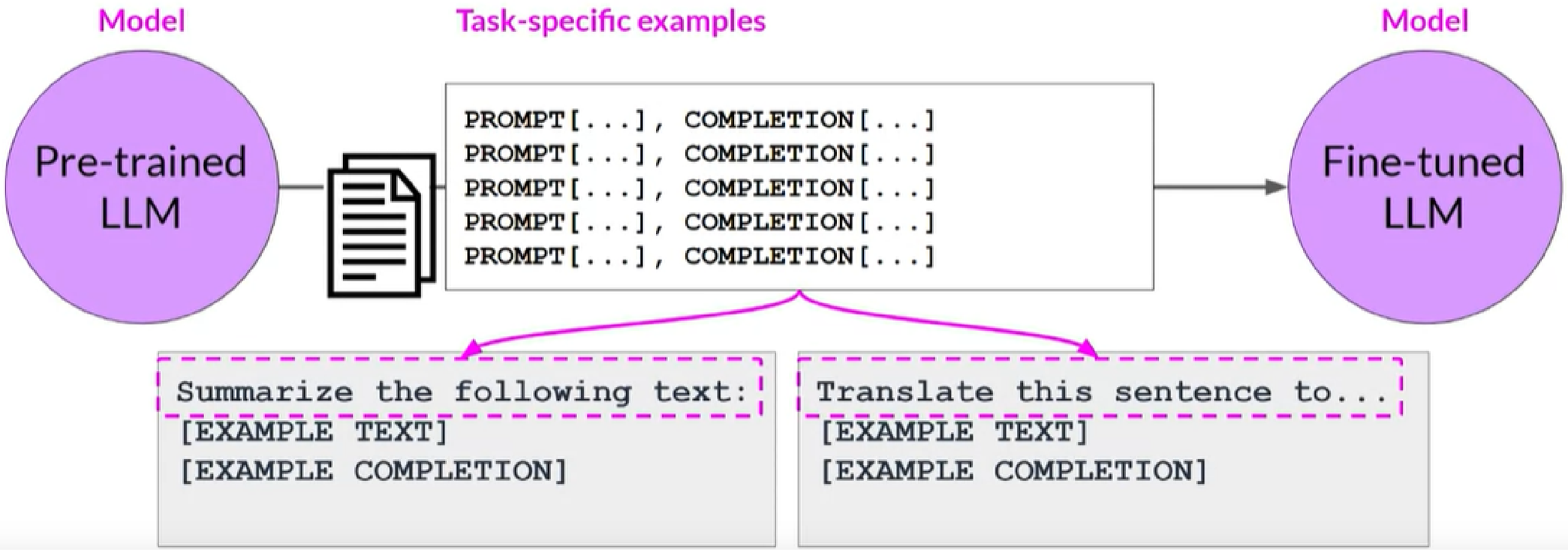
Fine Tuning

LLM fine-tuning at a high level

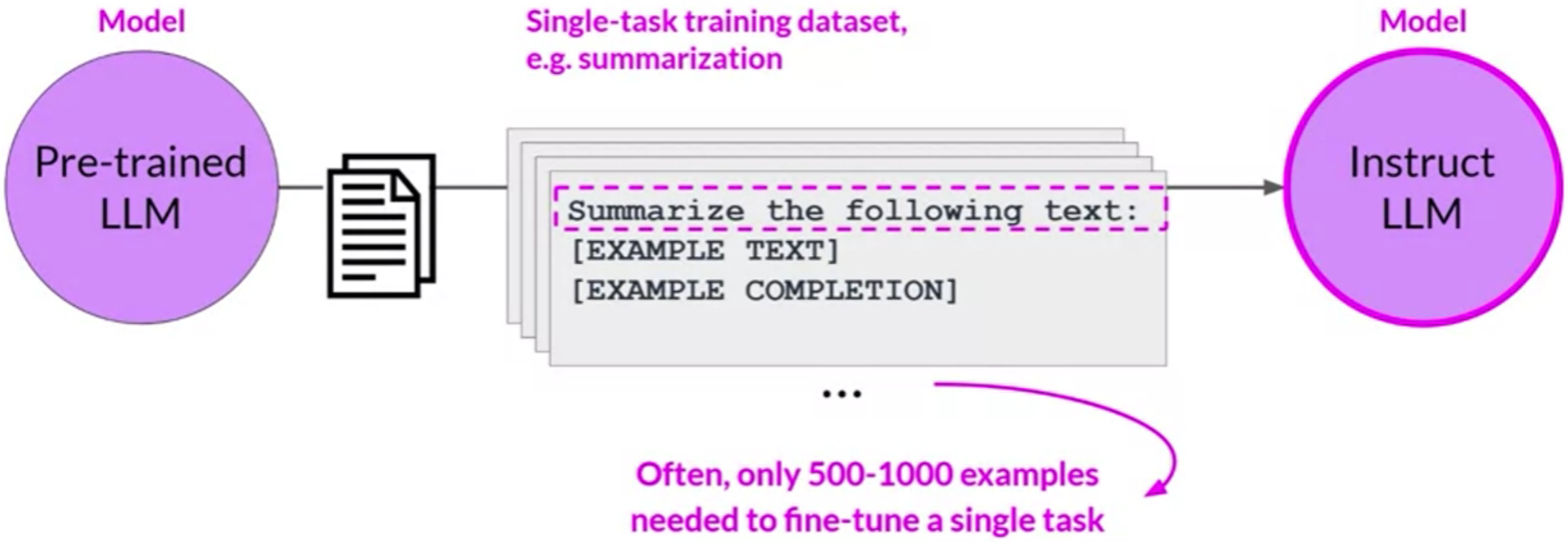


Using prompts to fine-tune LLMs with instruction

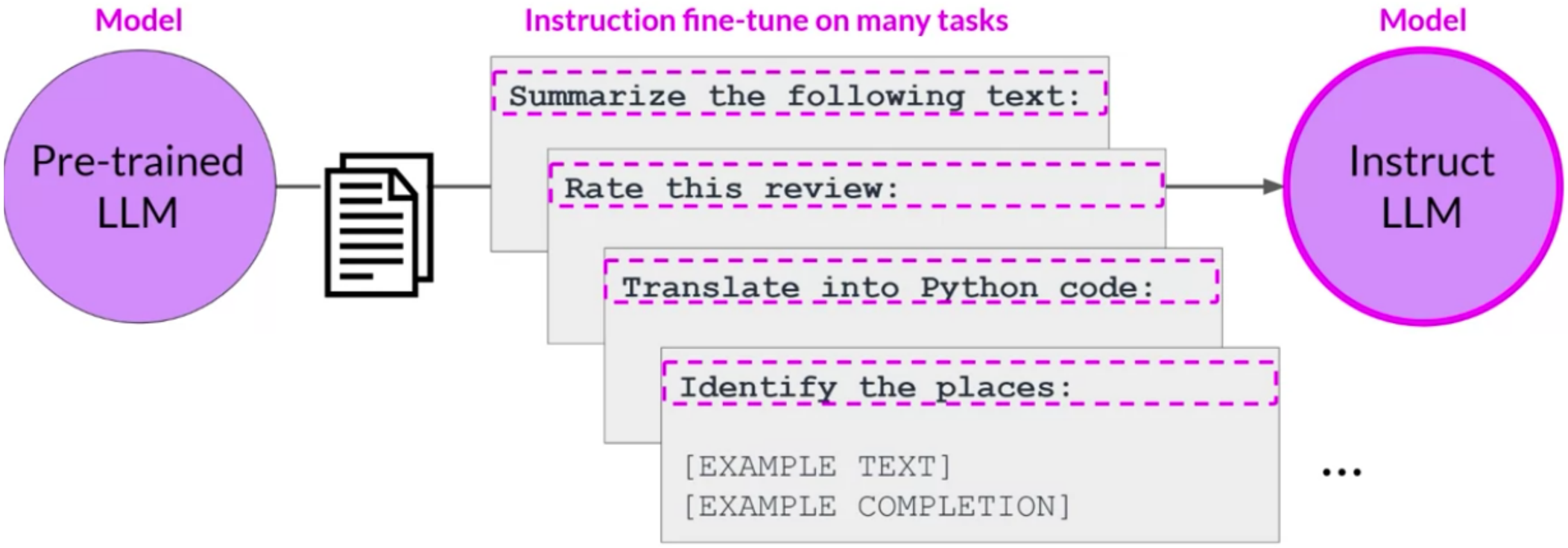
LLM fine-tuning



Fine-tuning on a single task



Multi-task, instruction fine-tuning



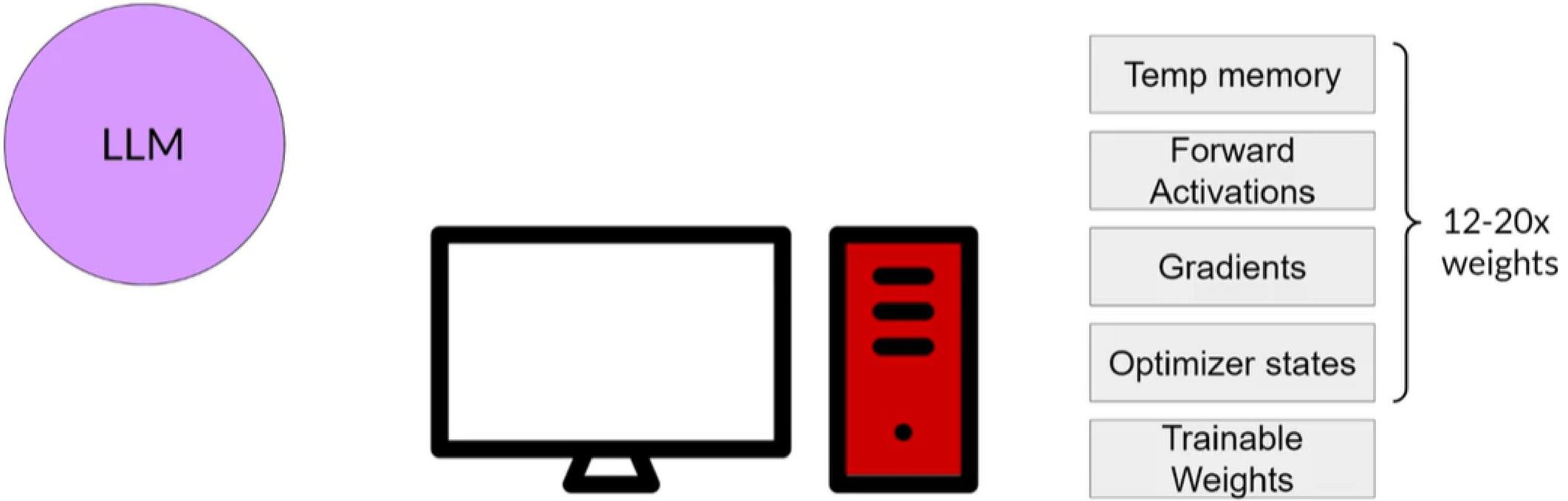
catastrophic forgetting, is the tendency of an artificial neural network to abruptly and drastically forget previously learned information upon learning new information.

How to avoid catastrophic forgetting

- First note that you might not have to!
- Fine-tune on **multiple tasks** at the same time
- Consider **Parameter Efficient Fine-tuning** (PEFT)

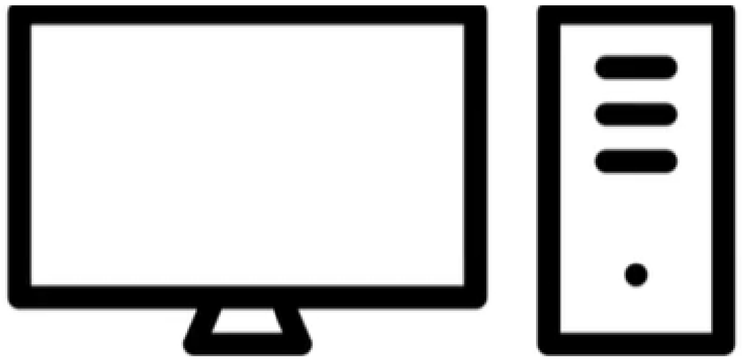
PEFT

Full fine-tuning of large LLMs is challenging



Parameter efficient fine-tuning (PEFT)

LLM

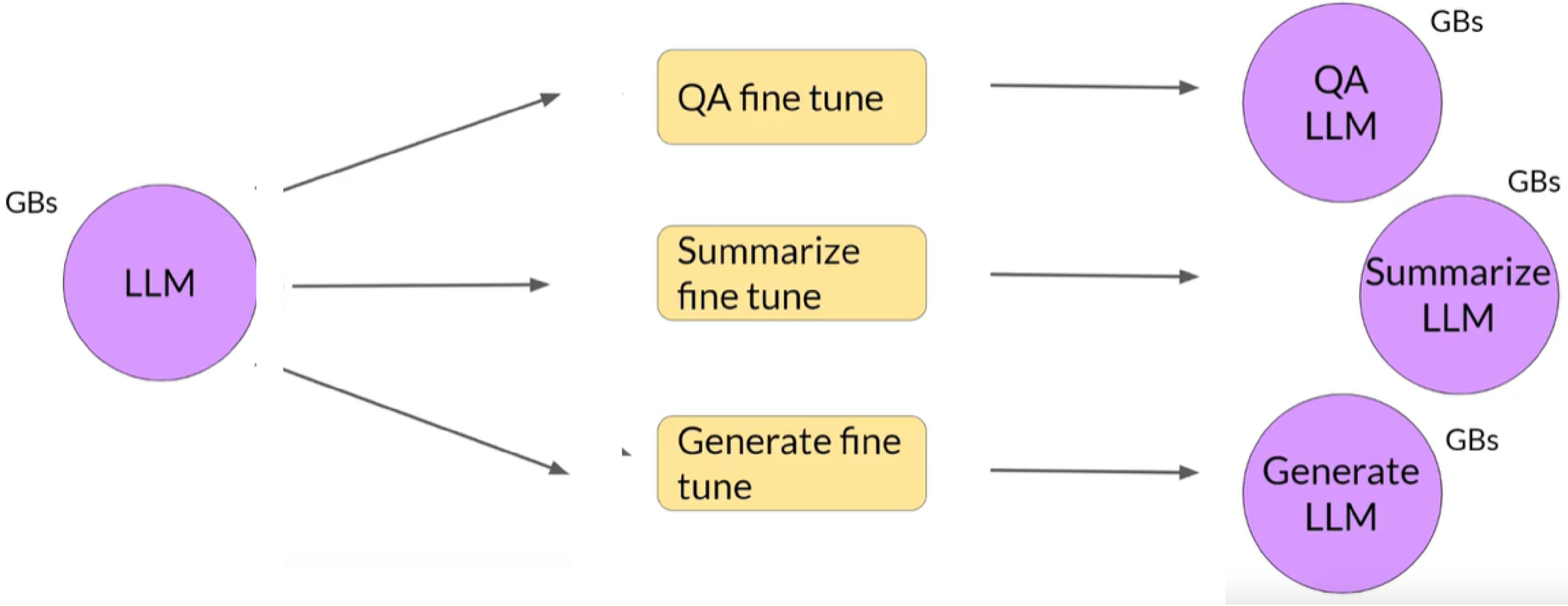


Less prone to
catastrophic forgetting

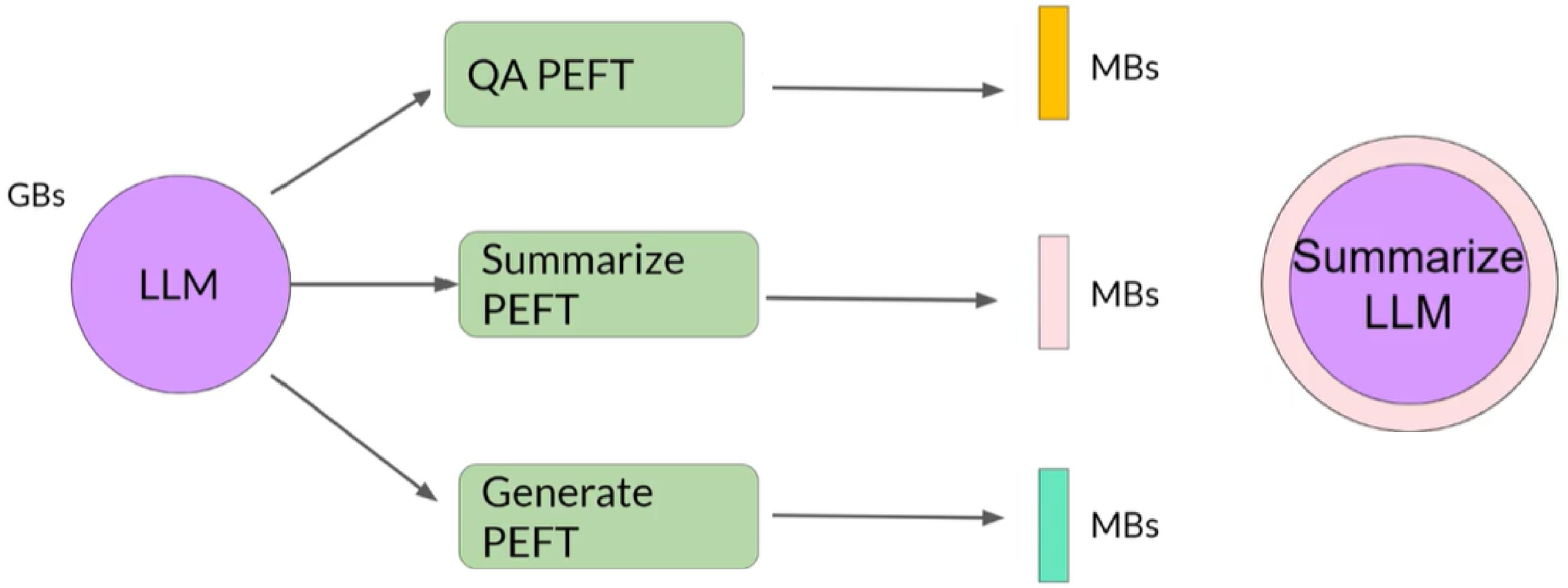
Other
components

Trainable
weights

Full fine-tuning creates full copy of original LLM per task



PEFT fine-tuning saves space and is flexible



PEFT methods

Selective

Select subset of initial LLM parameters to fine-tune

Reparameterization

Reparameterize model weights using a low-rank representation

LoRA

Additive

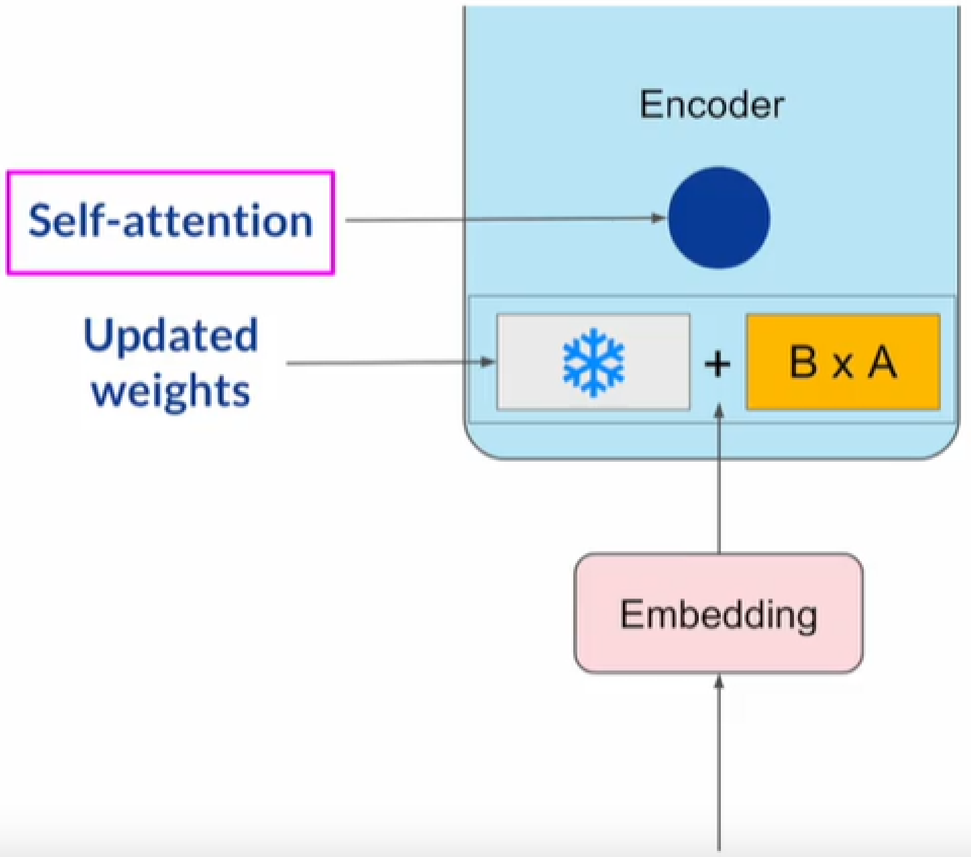
Add trainable layers or parameters to model

Adapters

Soft Prompts

Prompt Tuning

LoRA: Low Rank Adaption of LLMs



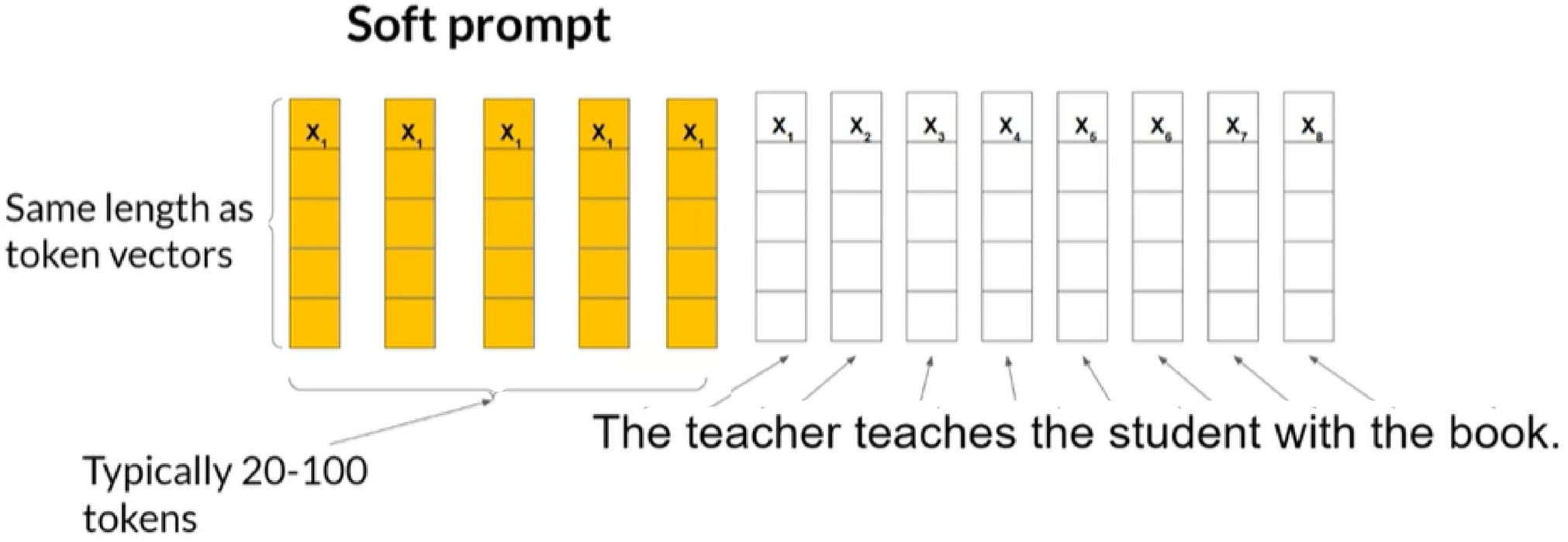
1. Freeze most of the original LLM weights.
2. Inject 2 **rank decomposition matrices**
3. Train the weights of the smaller matrices

Steps to update model for inference:

1. Matrix multiply the low rank matrices

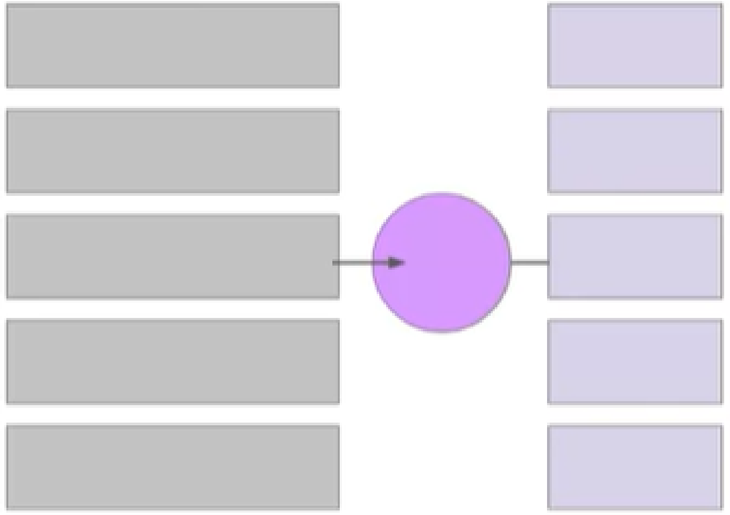
2. Add to original weights

Prompt tuning adds trainable “soft prompt” to inputs



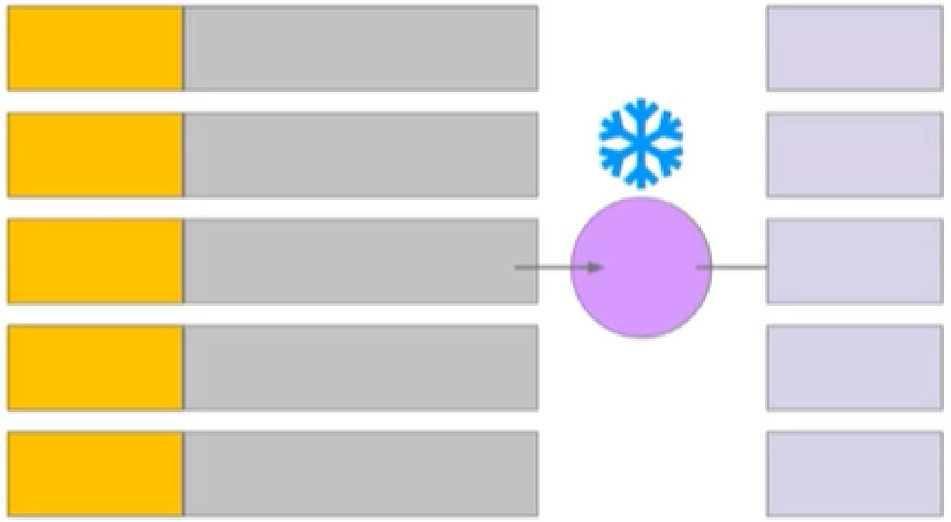
Full Fine-tuning vs prompt tuning

Weights of model updated during training



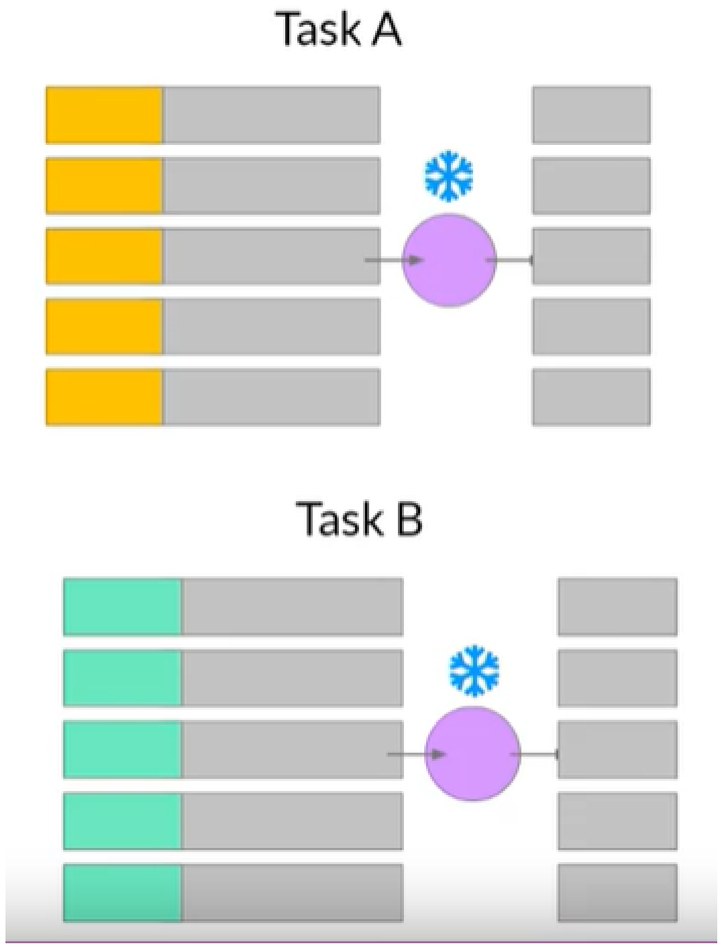
Millions to Billions of parameter updated

Weights of model frozen and soft prompt trained

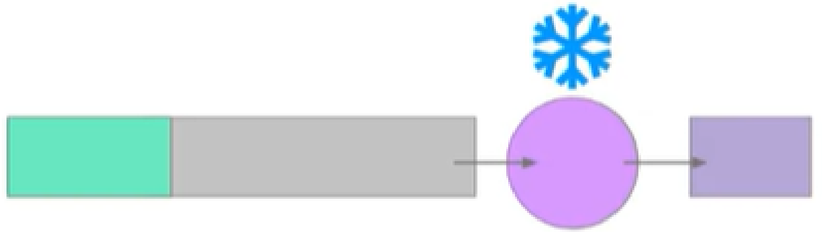


10K - 100K of parameters updated

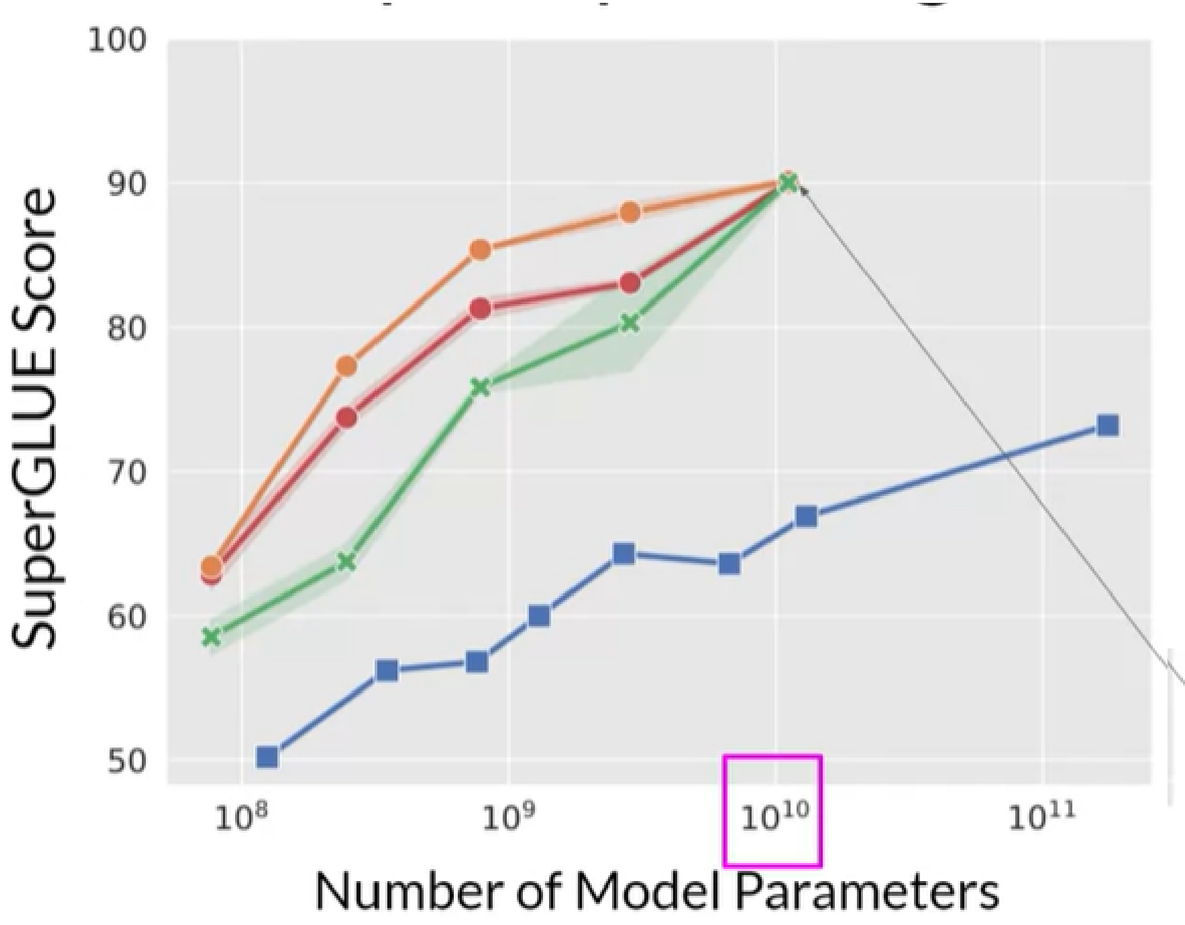
Prompt tuning for multiple tasks



Switch out soft prompt at inference time to change task!



Performance of prompt tuning



- Full Fine-tuning
- Multi-task Fine-tuning
- Prompt tuning
- Prompt engineering

Prompt tuning can be as effective as full Fine-tuning for

Model Evaluation

&

Benchmark

LLM Evaluation - Metrics

- Used for text summarization
- Compares a summary to one or more reference summaries

- Used for text translation
- Compares to human-generated translations

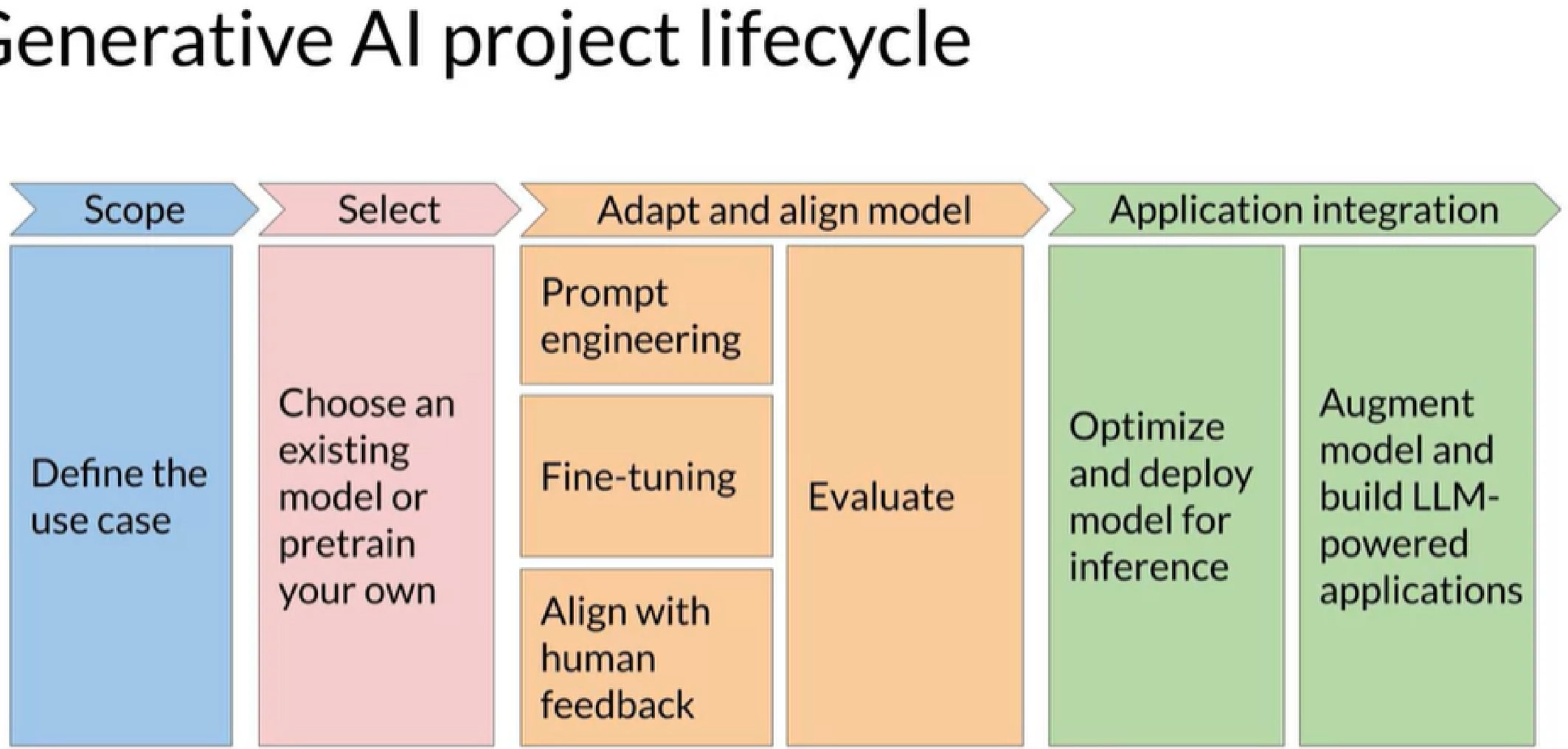
Evaluation benchmarks

MMLU (Massive Multitask Language Understanding)

BIG-bench

	Gemini Ultra	Gemini Pro	GPT-4	GPT-3.5	PaLM 2-L	Claude 2
MMLU Multiple-choice questions in 57 subjects (professional & academic) (Hendrycks et al., 2021a)	**90.04%** CoT@32*	79.13% CoT@8*	87.29% CoT@32 (via API**)	70% 5-shot	78.4% 5-shot	78.5% 5-shot CoT
	83.7% 5-shot	71.8% 5-shot	86.4% 5-shot (reported)			
GSM8K Grade-school math (Cobbe et al., 2021)	**94.4%** Maj1@32	86.5% Maj1@32	92.0% SFT & 5-shot CoT	57.1% 5-shot	80.0% 5-shot	88.0% 0-shot
MATH Math problems across 5 difficulty levels & 7 subdisciplines (Hendrycks et al., 2021b)	**53.2%** 4-shot	32.6% 4-shot	52.9% 4-shot (via API**)	34.1% 4-shot (via API**)	34.4% 4-shot	—
			50.3% (Zheng et al., 2023)			
BIG-Bench-Hard Subset of hard BIG-bench tasks written as CoT problems (Srivastava et al., 2022)	**83.6%** 3-shot	75.0% 3-shot	83.1% 3-shot (via API**)	66.6% 3-shot (via API**)	77.7% 3-shot	—
HumanEval Python coding tasks (Chen et al., 2021)	**74.4%** 0-shot (IT)	67.7% 0-shot (IT)	67.0% 0-shot (reported)	48.1% 0-shot	—	70.0% 0-shot
Natural2Code Python code generation. (New held-out set with no leakage on web)	**74.9%** 0-shot	69.6% 0-shot	73.9% 0-shot (via API**)	62.3% 0-shot (via API**)	—	—
DROP Reading comprehension & arithmetic. (metric: F1-score) (Dua et al., 2019)	**82.4** Variable shots	74.1 Variable shots	80.9 3-shot (reported)	64.1 3-shot	82.0 Variable shots	—
HellaSwag (validation set) Common-sense multiple choice questions (Zellers et al., 2019)	87.8% 10-shot	84.7% 10-shot	**95.3%** 10-shot (reported)	85.5% 10-shot	86.8% 10-shot	—
WMT23 Machine translation (metric: BLEURT) (Tom et al., 2023)	**74.4** 1-shot (IT)	71.7 1-shot	73.8 1-shot (via API**)	—	72.7 1-shot	—

Reinforcement Learning & Gen AI Project Lifecycle



Reinforcement Learning

Reinforcement learning (RL) is a type of machine learning paradigm where an agent learns to make decisions by interacting with an environment. The agent takes actions in the environment and receives feedback in the form of rewards or penalties, which indicates the quality of its actions

Why Reinforcement Learning

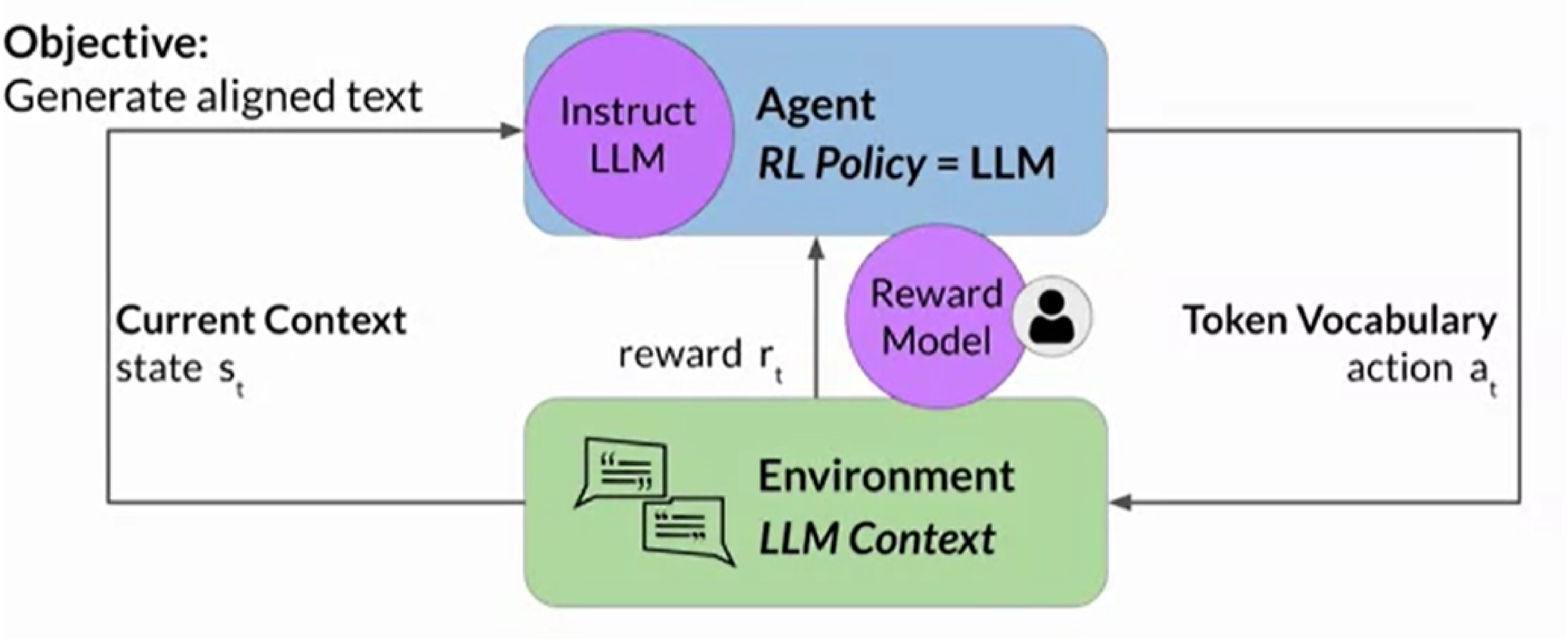
model behaves badly :

- Toxic Language
- Aggressive Response
- Providing Dangerous Information
- Incorrect Answer
- Offensive , Discriminatory or eliciting criminal behaviour

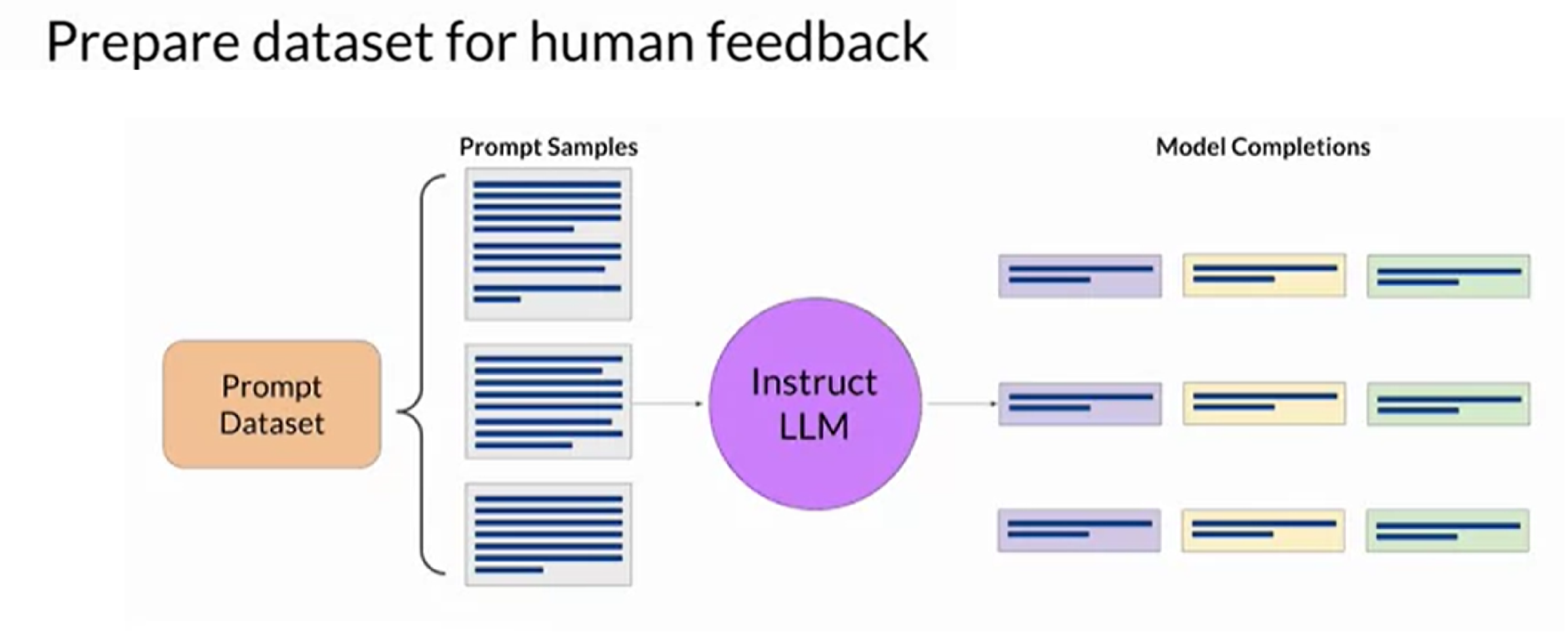
RLHF : Reinforcement Learning From Human Feedback

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

Reinforcement learning: fine-tune LLMs

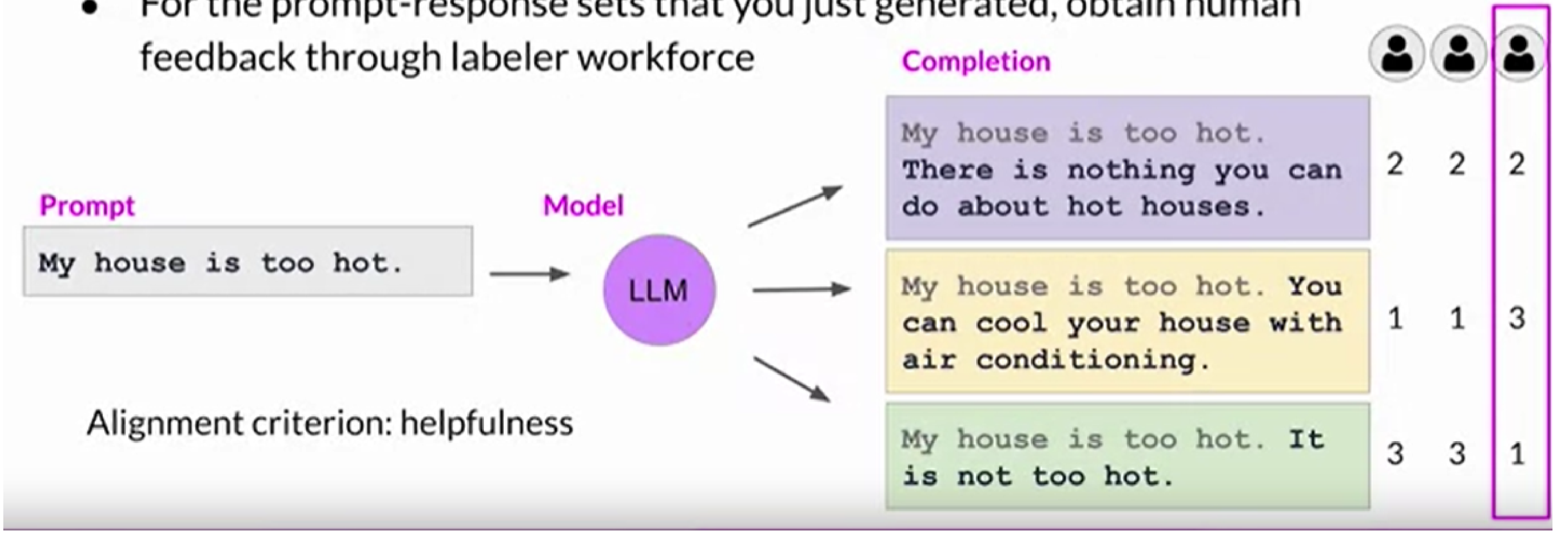


First steip in Fine-tuning an LLM with RLHF is to select a model to work with and use it to prepare a dataset for human feedback



Collect human feedback

- Define your model alignment criterion
- For the prompt-response sets that you just generated, obtain human feedback through labeler workforce



Sample instructions for human labelers

* 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.

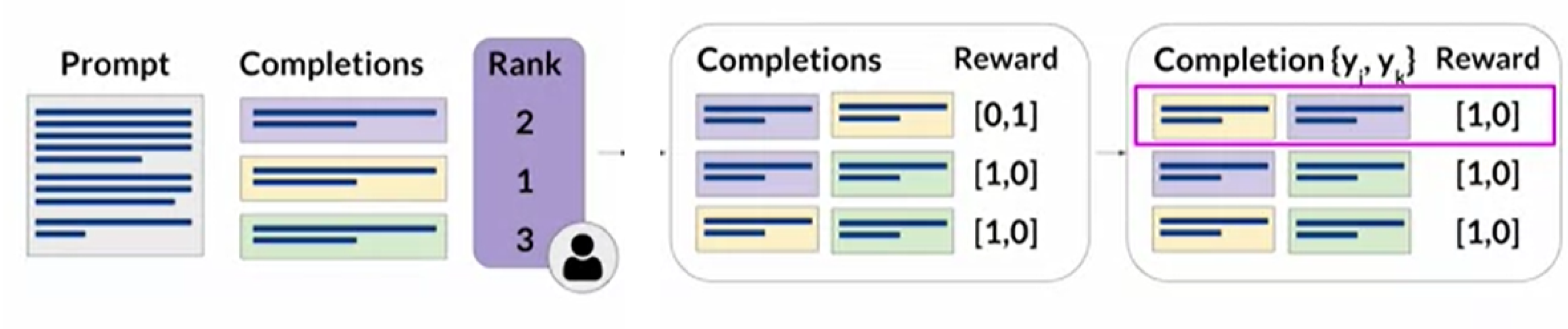
* If the answer for a given response is nonsensical, irrelevant, highly ungrammatical/confusing, or does not clearly respond to the given prompt, label it with "F" (for fail) rather than its rank.

* 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.

Reward Model :

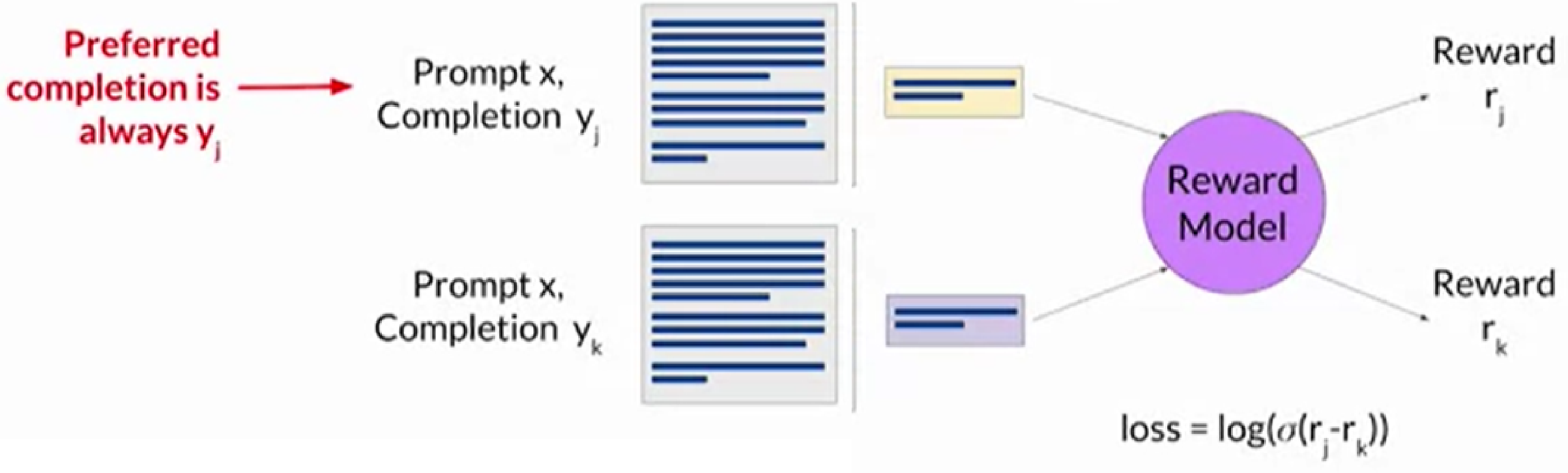
Prepare labeled data for training

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



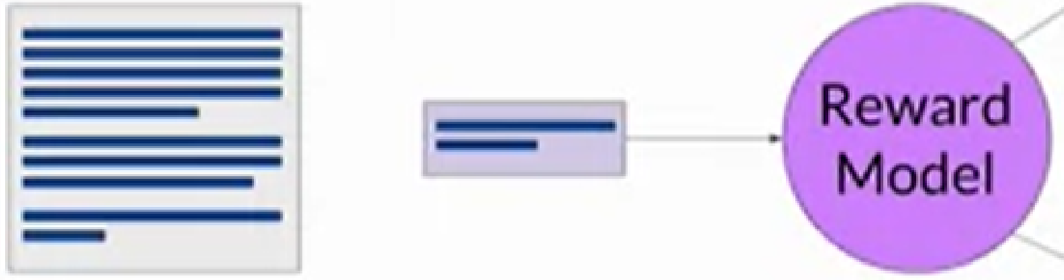
Train reward model

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



Use the reward model

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

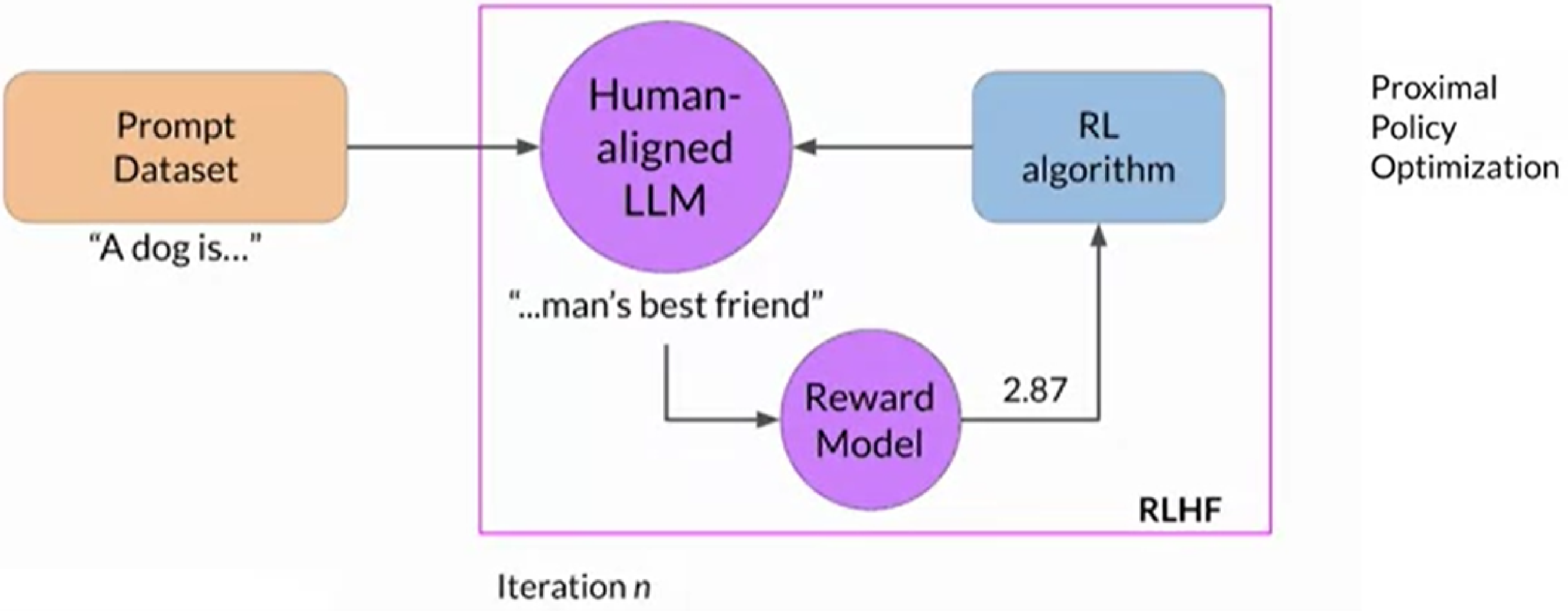


Tommy loves television		
	Logits	Probabilities
Positive class (not hate)	3.171875	0.996093
Negative class (hate)	-2.609375	0.003082

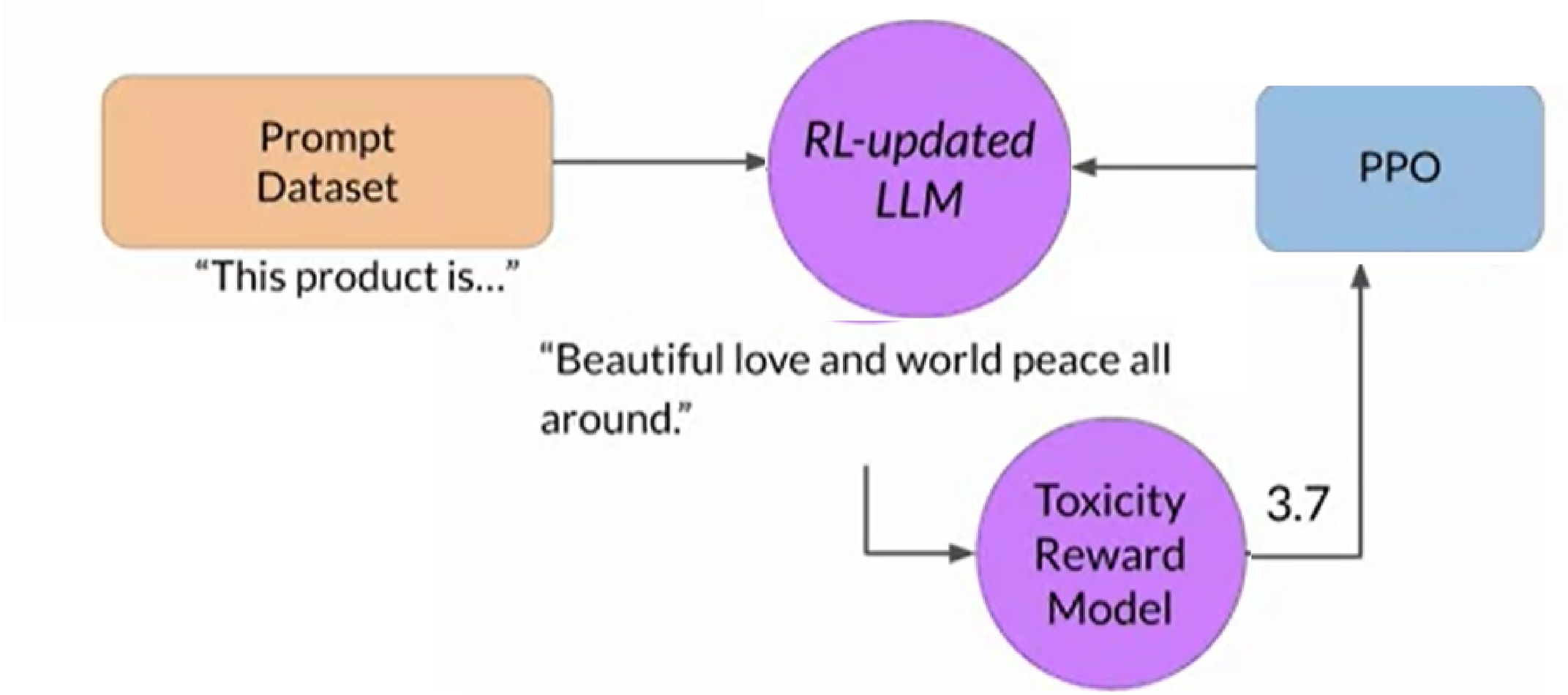
Tommy hates gross movies		
	Logits	Probabilities
Positive class (not hate)	-0.535156	0.337890
Negative class (hate)	0.137695	0.664062

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

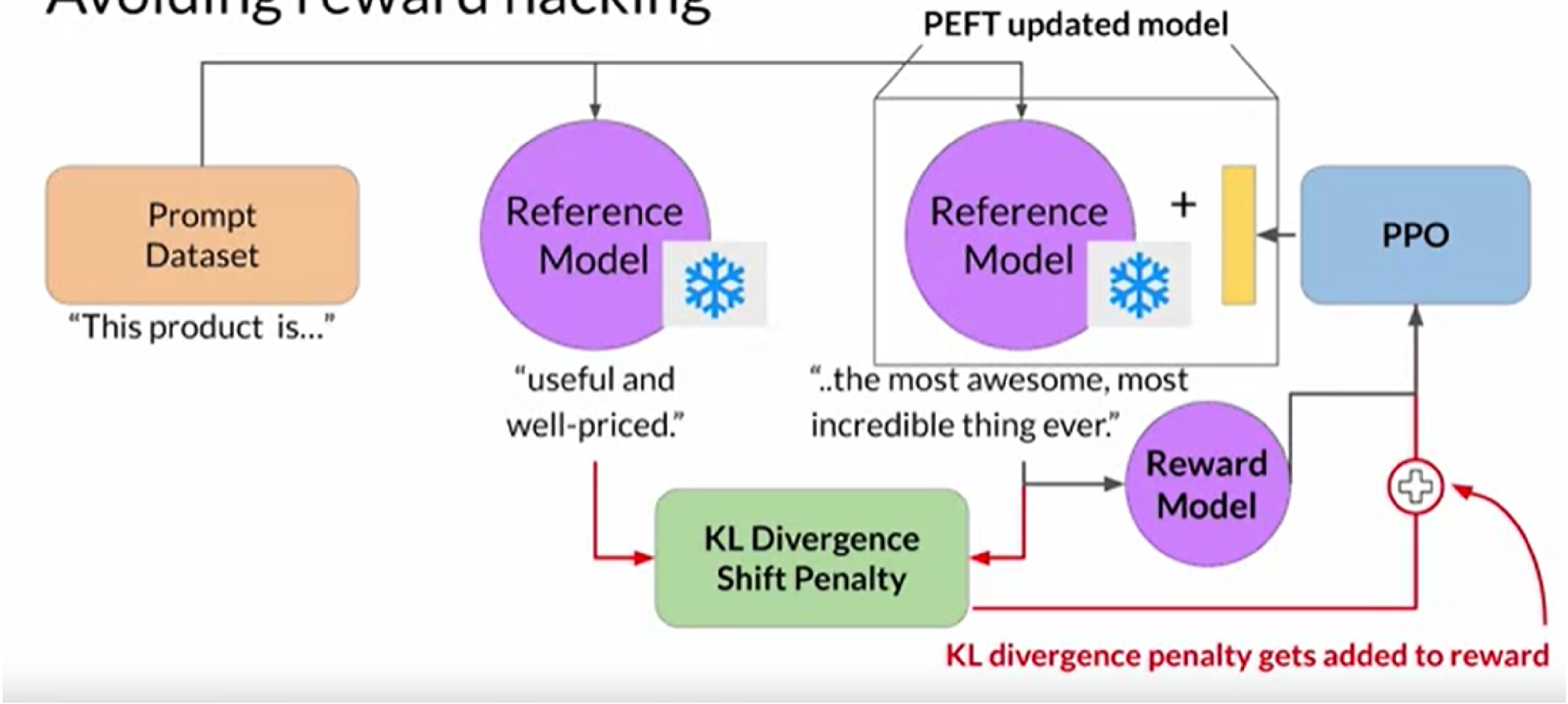
Use the reward model to fine-tune LLM with RL



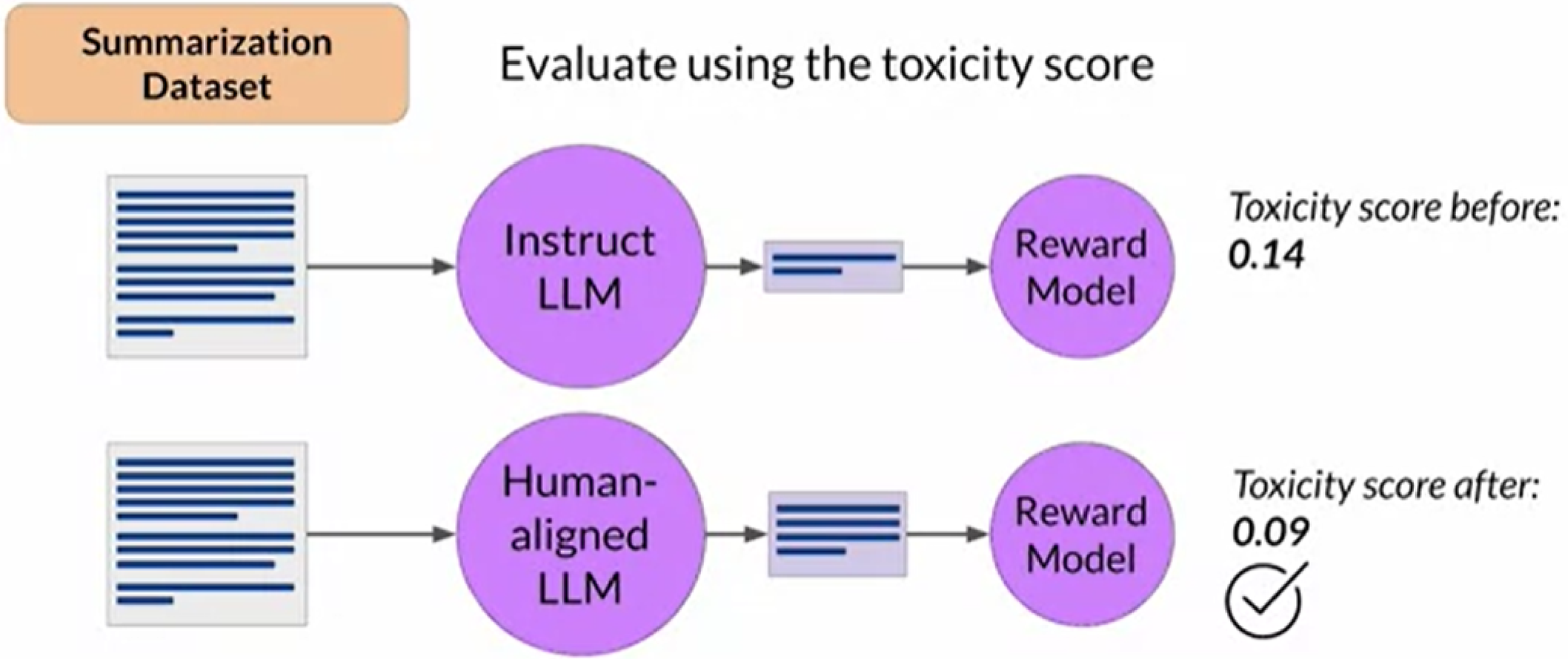
Potential problem: reward hacking



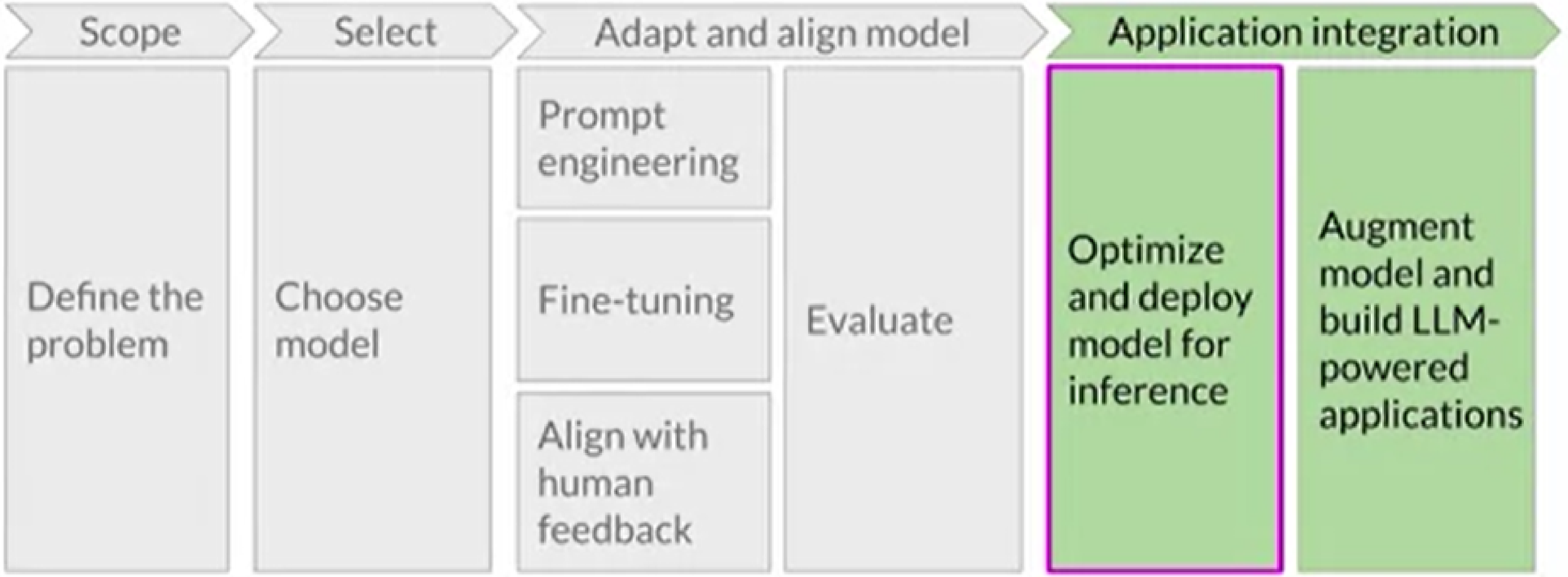
Avoiding reward hacking



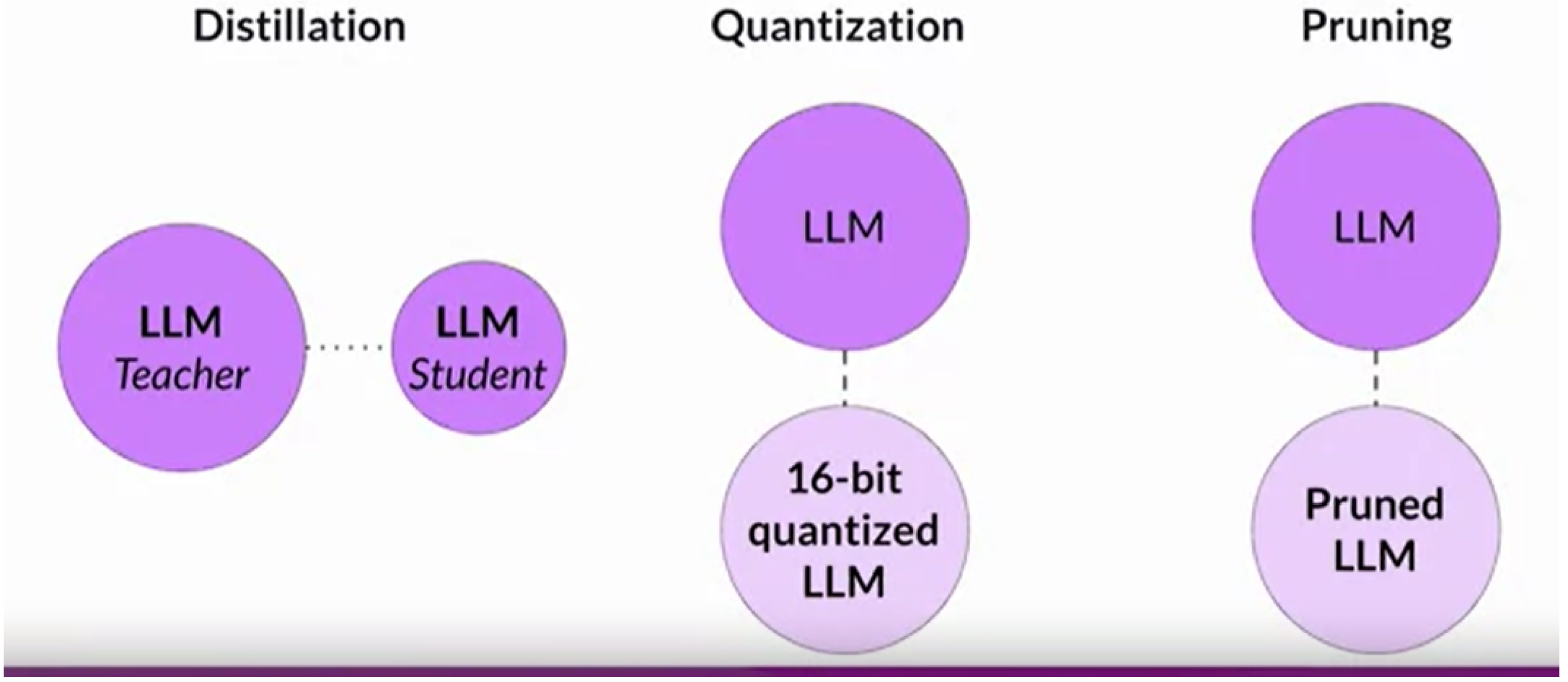
Evaluate the human-aligned LLM



Generative AI project lifecycle



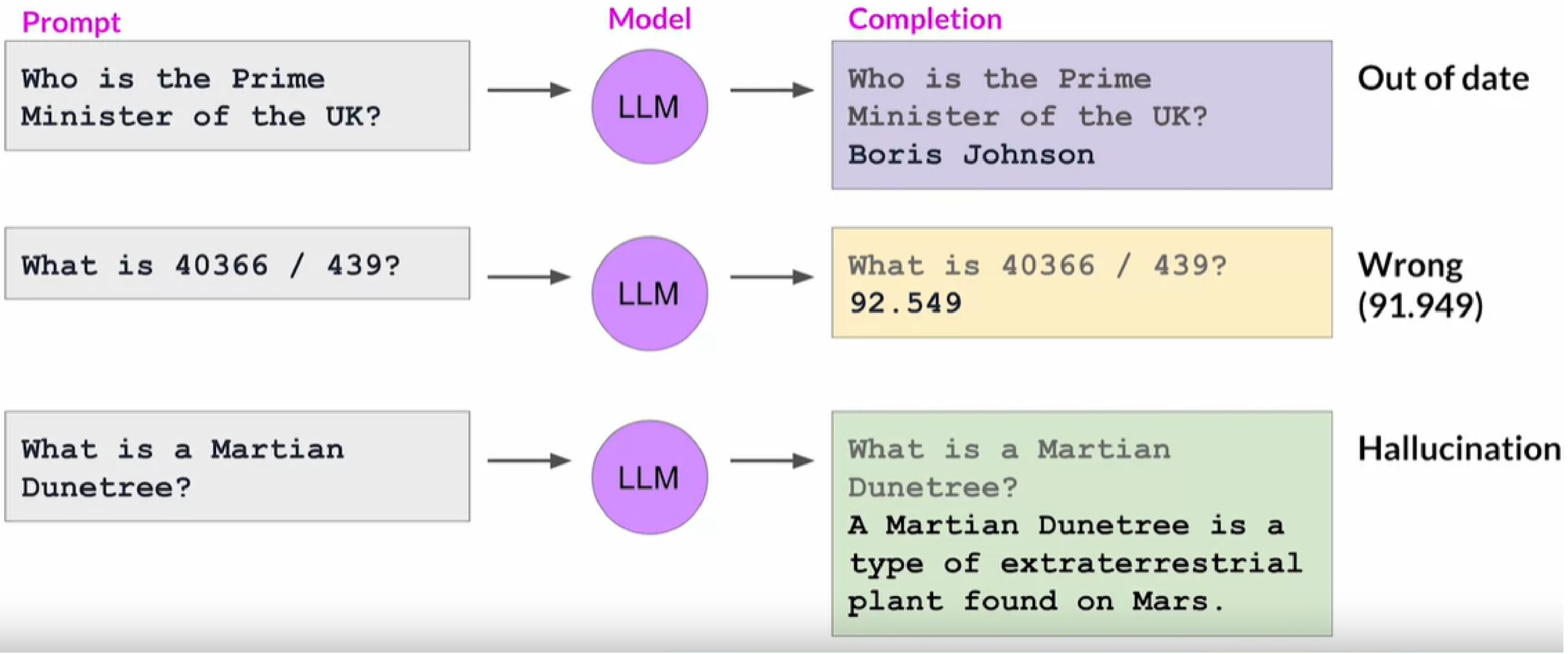
LLM optimization techniques



Cheat Sheet - Time and effort in the lifecycle

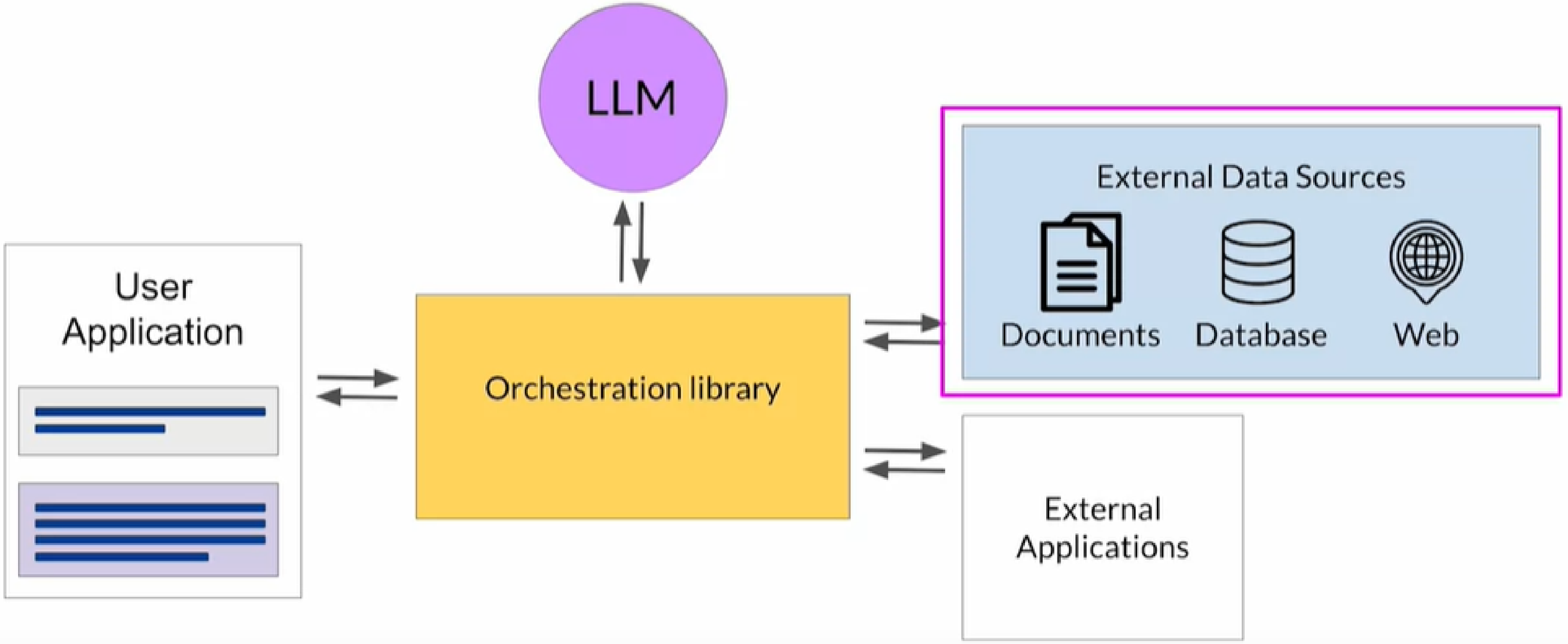
	Pre-training	Prompt engineering	Prompt tuning and fine-tuning	Reinforcement learning/human feedback	Compression/ optimization/ deployment
Training duration	Days to weeks to months	Not required	Minutes to hours	Minutes to hours similar to fine-tuning	Minutes to hours
Customization	Determine model architecture, 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

Models having difficulty

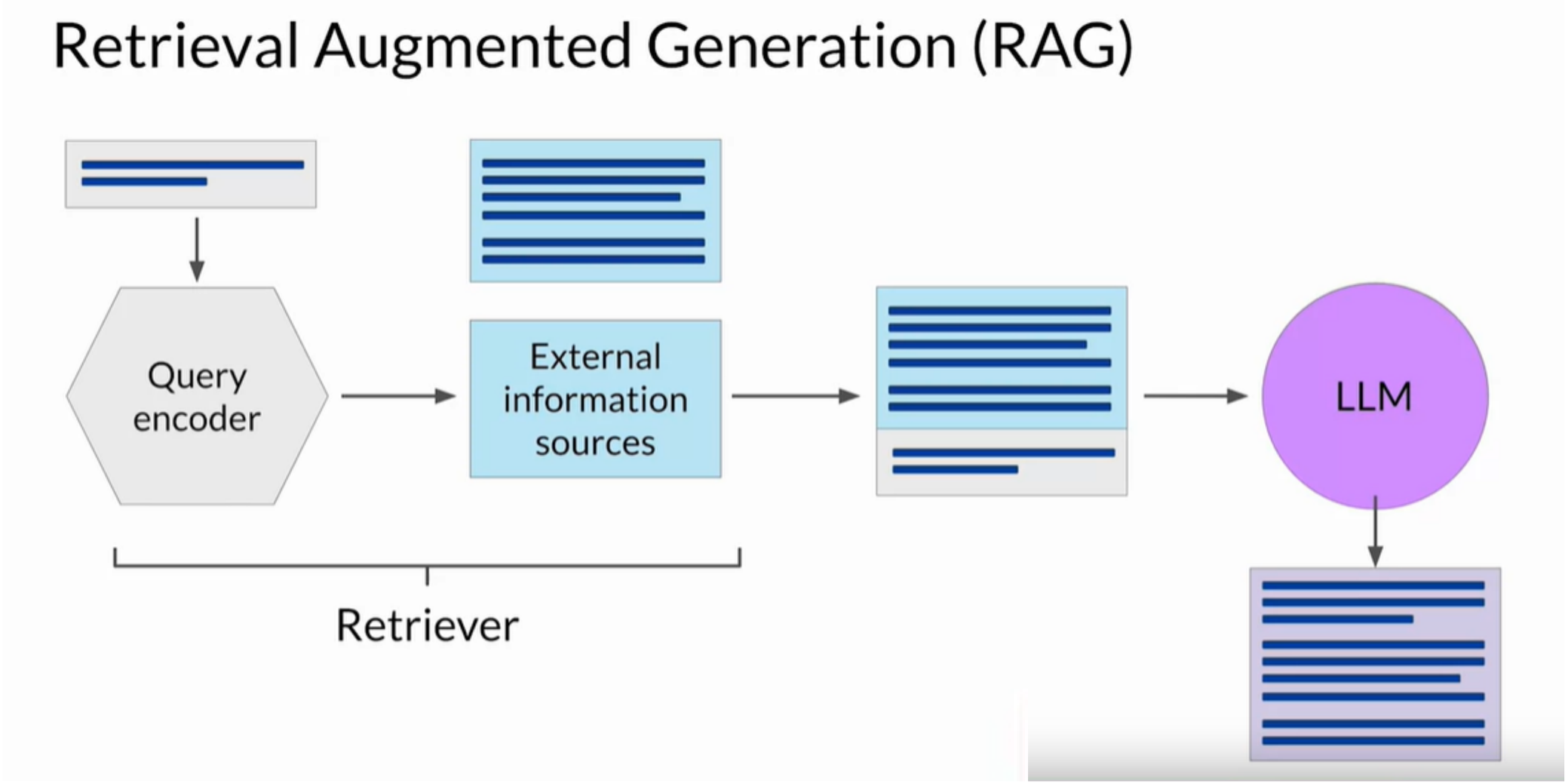


Generative AI project lifecycle

LLM-powered applications



RAG is a framework for providing LLMs access to data they did not see during training.



Large Language Models

BCV

Application Layer

Copywriting
Jasper
copy.ai
Headlime
co:here
HyperWrite
Writesonic
Contenda
unbounce
copysmith

Coding
tabnine
MUTABLE AI
Codiga
co:here
GitHub Copilot
CODEGEN

Dev Tools
algolia
warp
Mintlify
cogram
Debuild
repl.it

Chat / Comms
MessageBird
Replier.ai
Sapling
FABLE

BizOps
viable
Enterpret
tabulate
Anecdote
OTHERSIDE AI
casetext
Dover

Infrastructure Layer

Model Creation
AI21 labs
OpenAI
NVIDIA
Adept
EleutherAI
ANTHROPIC
Google AI

Hardware
habana
NVIDIA
SambaNova SYSTEMS
cerebras
GRAPHCORE

Fine Tuning
Google AI
aws
OpenAI
Hugging Face

Inference
OpenAI
Hugging Face

Foundation Large Language Model Stack

End User Applications

Writing Assistant

POSTLY, Scalenut, copymatic, Spellbook by Rally, EmailMagic AI, Reply, AISEO, supertranslate, Peppertype.ai, Kwirk, MateAI, Simplified, LanguageTool Your writing assistant, Writesonic, Rytr, Soofy., anyword, HirePeople, grammarly, ChatGPT Writer, copy.ai, Writey A.I, NeuroSpell

Content & Idea Creation

kickresume, Text Generator Plugin, InstaNovel, Sivi, LEGALESE DECODER, text.cortex, SEO.ai, PatentPal, Paraphrase Tool, contents.com, Jasper, tome, daydrm.ai, CSM

Generative & Search Assistants

Certainly., verloop.io, nlx KOMO, Perplexity, Dystr, Lexii.ai, Landbot, ExplainThis, ProductBot, MightyGPT, EvelynAI, Andi, AI Alfred, Moveworks, Teach Anything

Data Extraction & Conversational Search

Channel, kadoa, context, RTutor, OSS Insight, Corpora, Elicit, TableTalk

Developer/ Coding Assistants

Refraction, phind, CodeAssist, CodeGeeX

LLM Development Tools

LLM API Build Frameworks

DUST, PromptChainer, LangChain, Steamship, stack, Chatbase, Humanloop, re:tune, DeepPavlov, haystack by deepset, ChatBotKit, Fixie.ai, NLP Cloud, auto TRAIN

Prompt Engineering Tools

prisms, vellum, Everyprompt, PROMPT[STACKS], Dyno Prompt Engineering IDE, Promptify, Spellbook, vellum, Baseplate

Data Centric Tooling

HumanFirst, scale, Snorkel

Foundation Models & Hubs

Models & Hubs

Foundation LLM Models

OpenAI, Goose AI, LaMD, Cohere, AI21labs, PaLM

Open-Sourced LLM Models

Bloom, Whisper, EleutherAI

Code Models

Codex, BigCode

Model Hubs

Hugging Face, GitHub

Cobus Greyling
https://www.linkedin.com/in/cobusgreyling

Version 3.0

Quiz Time