Advanced Methods in Natural Language Processing

Session 9: LLMs, Prompting, RAG & Fine-Tuning

Arnault Gombert

June 2026

Barcelona School of Economics

Introduction

Introduction to Today's Lecture on LLMs and ChatGPT

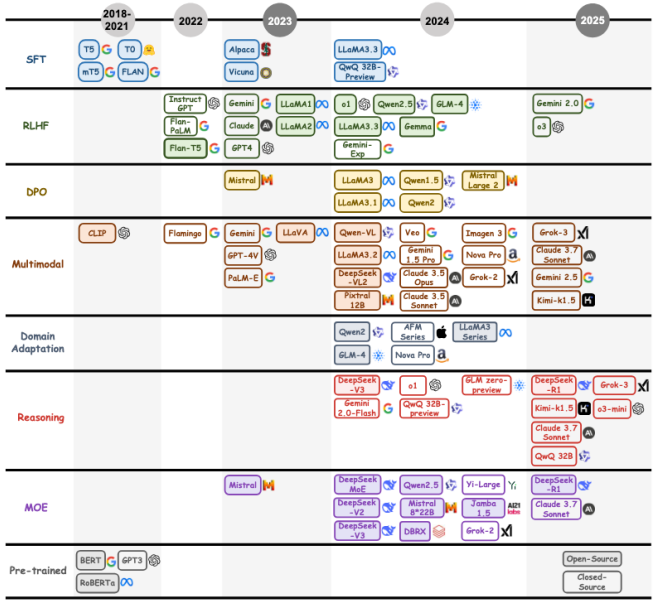
Today, we delve into the world of LLMs such as ChatGPT, exploring their advancements, applications, and the intricacies of prompt engineering, fine-tuning, and retrieval-augmented generation (RAG).

Session Overview:

- **Overview of LLMs:** Introducing text-only and multimodal models, and their evolution since 2022.
- **Main Use Cases:** Exploring the diverse applications of LLMs in various domains.
- **Prompt Engineering:** Understanding the art of effectively communicating with LLMs to achieve desired outcomes.
- **Retrieval-Augmented Generation (RAG):** Leveraging external knowledge bases to enhance LLMs' responses.
- **Fine-Tuning Techniques:** Techniques to customize LLMs for specific tasks or datasets.

Training of ChatGPT: Process and Innovations

- **Foundation Model**: ChatGPT builds upon a large transformer-based language model, similar to GPT-3, trained on a diverse range of internet text.

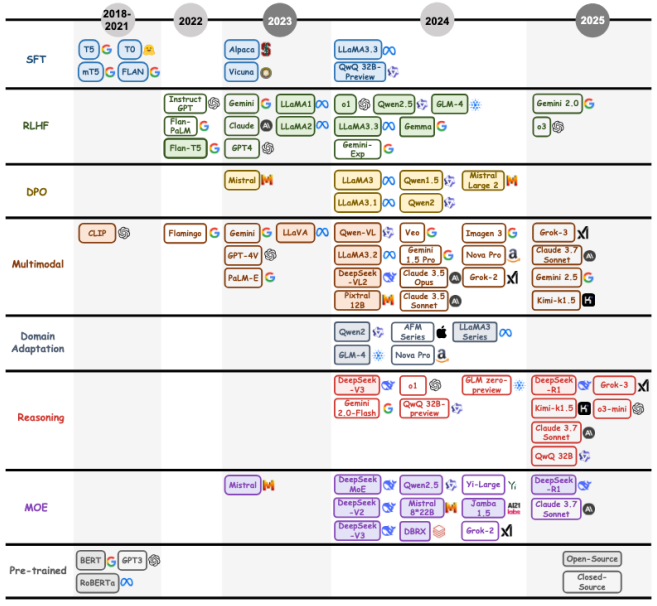


Evolution of Texts LLM

Tie et al. (2025)

Training of ChatGPT: Process and Innovations

- **Foundation Model**: ChatGPT builds upon a large transformer-based language model, similar to GPT-3, trained on a diverse range of internet text.
- **Reinforcement Learning from Human Feedback (RLHF)**:

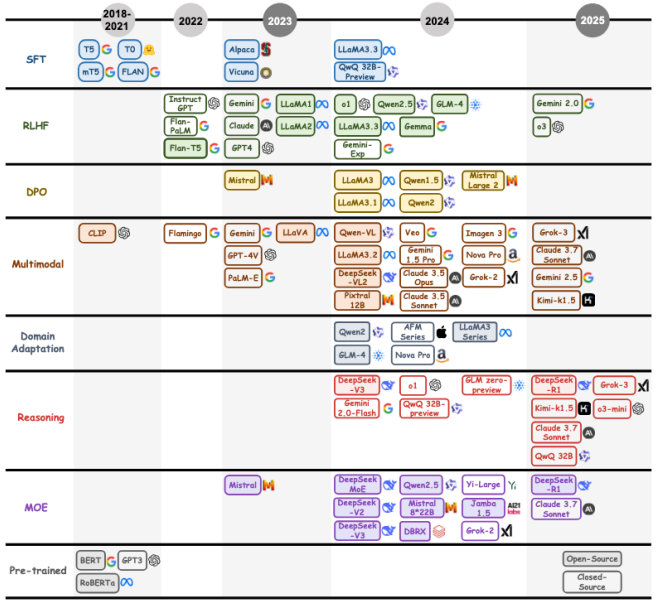


Evolution of Texts LLM

Tie et al. (2025)

Training of ChatGPT: Process and Innovations

- **Foundation Model**: ChatGPT builds upon a large transformer-based language model, similar to GPT-3, trained on a diverse range of internet text.
- **Reinforcement Learning from Human Feedback (RLHF)**:
  - *Supervised Fine-Tuning (SFT)*: Initial fine-tuning on a dataset of conversational prompts and responses.

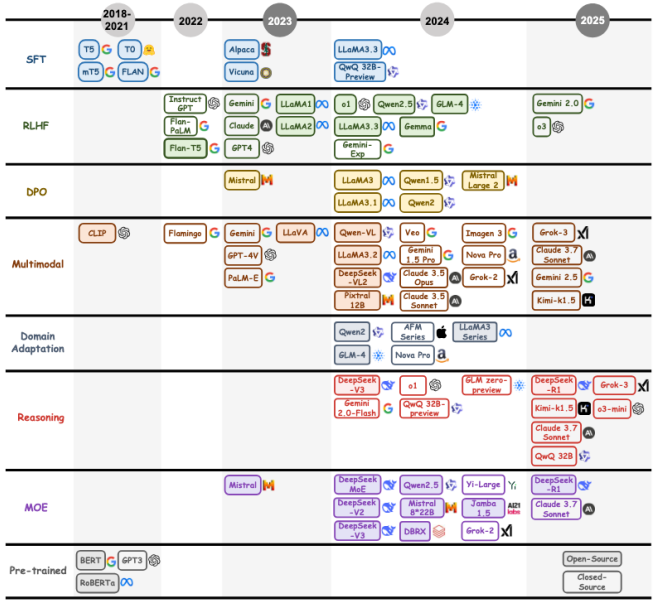


Evolution of Texts LLM

Tie et al. (2025)

Training of ChatGPT: Process and Innovations

- **Foundation Model**: ChatGPT builds upon a large transformer-based language model, similar to GPT-3, trained on a diverse range of internet text.
- **Reinforcement Learning from Human Feedback (RLHF)**:
  - *Supervised Fine-Tuning (SFT)*: Initial fine-tuning on a dataset of conversational prompts and responses.
  - *Reward Modeling (RM)*: Trained a reward model to predict scores given by human trainers for model-generated responses.

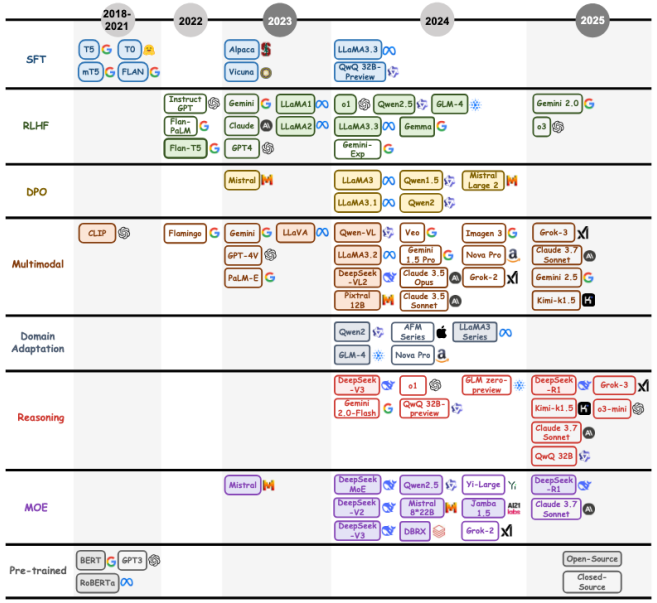


Evolution of Texts LLM

Tie et al. (2025)

Training of ChatGPT: Process and Innovations

- **Foundation Model**: ChatGPT builds upon a large transformer-based language model, similar to GPT-3, trained on a diverse range of internet text.
- **Reinforcement Learning from Human Feedback (RLHF)**:
  - *Supervised Fine-Tuning (SFT)*: Initial fine-tuning on a dataset of conversational prompts and responses.
  - *Reward Modeling (RM)*: Trained a reward model to predict scores given by human trainers for model-generated responses.
  - *Proximal Policy Optimization (PPO)*: Final fine-tuning phase using the reward model to guide training toward human preferences.

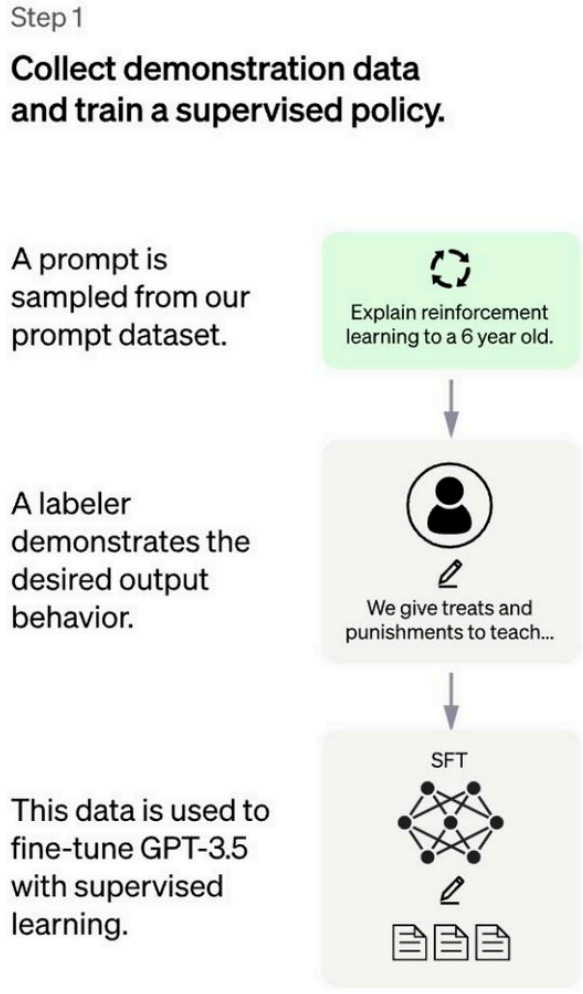


Evolution of Texts LLM

Tie et al. (2025)

Supervised Fine-Tuning (SFT) in ChatGPT

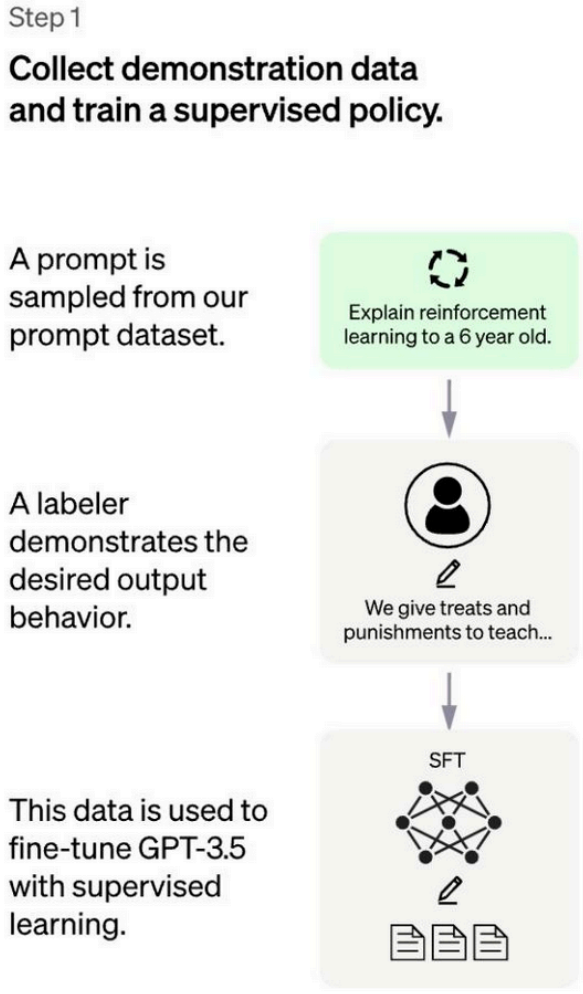
- **Objective**: Refine the foundational language model towards conversational understanding and response generation.



Supervised Fine-Tuning Phase

Supervised Fine-Tuning (SFT) in ChatGPT

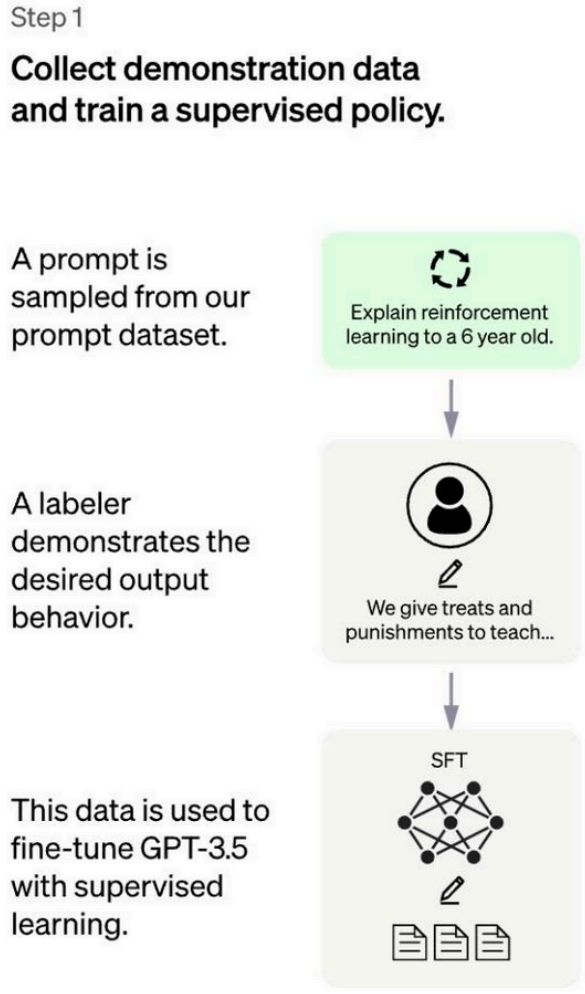
- **Objective**: Refine the foundational language model towards conversational understanding and response generation.
- **Process**:



Supervised Fine-Tuning Phase

Supervised Fine-Tuning (SFT) in ChatGPT

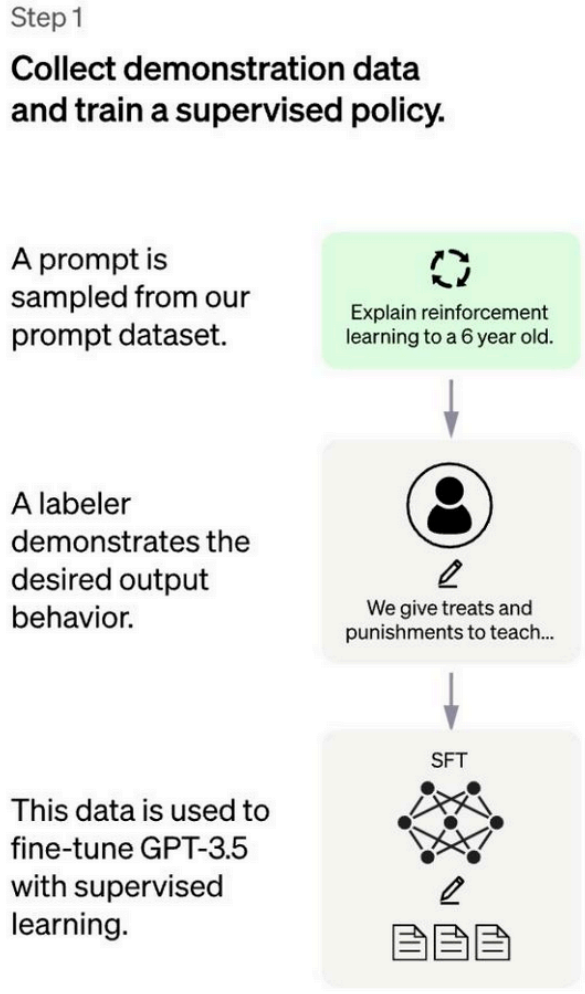
- **Objective**: Refine the foundational language model towards conversational understanding and response generation.
- **Process**:
  - Utilizes a curated dataset comprising diverse conversational prompts and corresponding human-written responses.



Supervised Fine-Tuning Phase

Supervised Fine-Tuning (SFT) in ChatGPT

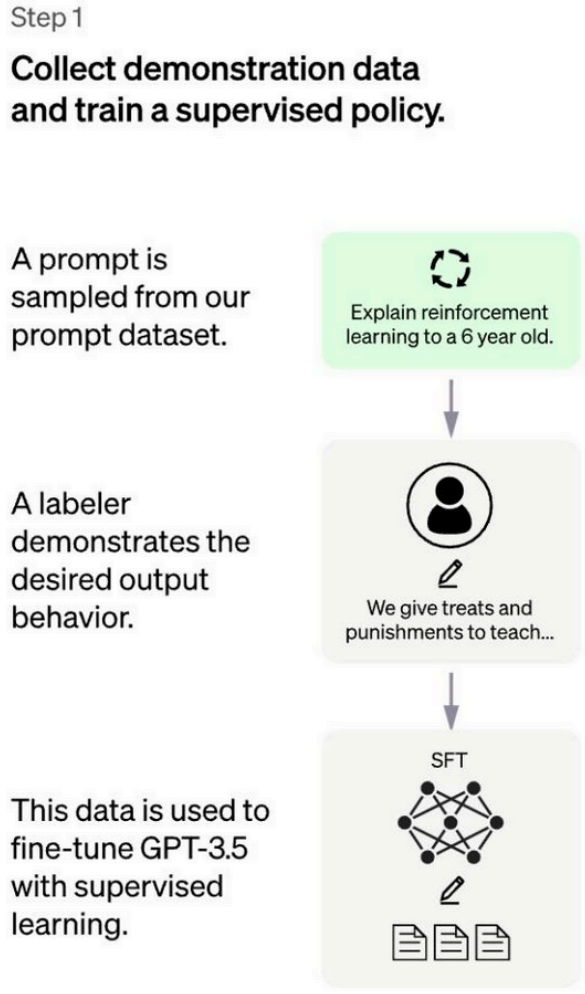
- **Objective**: Refine the foundational language model towards conversational understanding and response generation.
- **Process**:
  - Utilizes a curated dataset comprising diverse conversational prompts and corresponding human-written responses.
  - The model is fine-tuned to predict these responses accurately, aligning its outputs more closely with human-like conversational patterns.



Supervised Fine-Tuning Phase

Supervised Fine-Tuning (SFT) in ChatGPT

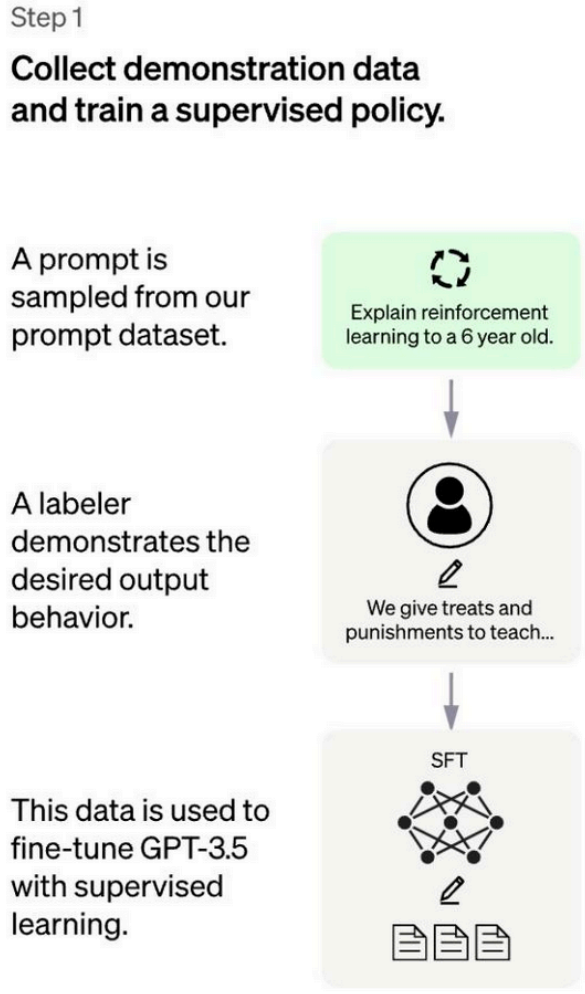
- **Objective**: Refine the foundational language model towards conversational understanding and response generation.
- **Process**:
  - Utilizes a curated dataset comprising diverse conversational prompts and corresponding human-written responses.
  - The model is fine-tuned to predict these responses accurately, aligning its outputs more closely with human-like conversational patterns.
- **Example**:



Supervised Fine-Tuning Phase

Supervised Fine-Tuning (SFT) in ChatGPT

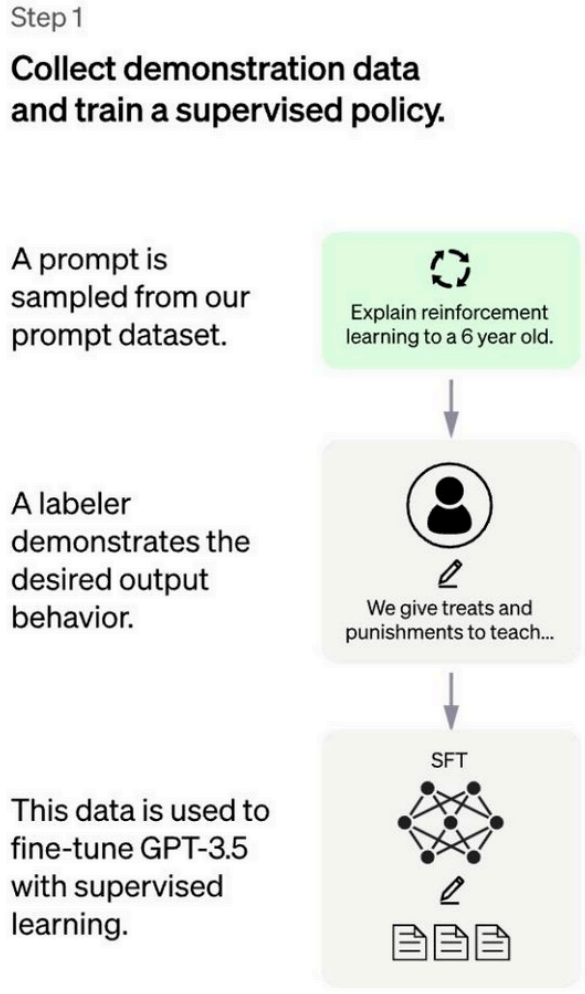
- **Objective**: Refine the foundational language model towards conversational understanding and response generation.
- **Process**:
  - Utilizes a curated dataset comprising diverse conversational prompts and corresponding human-written responses.
  - The model is fine-tuned to predict these responses accurately, aligning its outputs more closely with human-like conversational patterns.
- **Example**:
  - Prompt: "What's your favorite book and why?"



Supervised Fine-Tuning Phase

Supervised Fine-Tuning (SFT) in ChatGPT

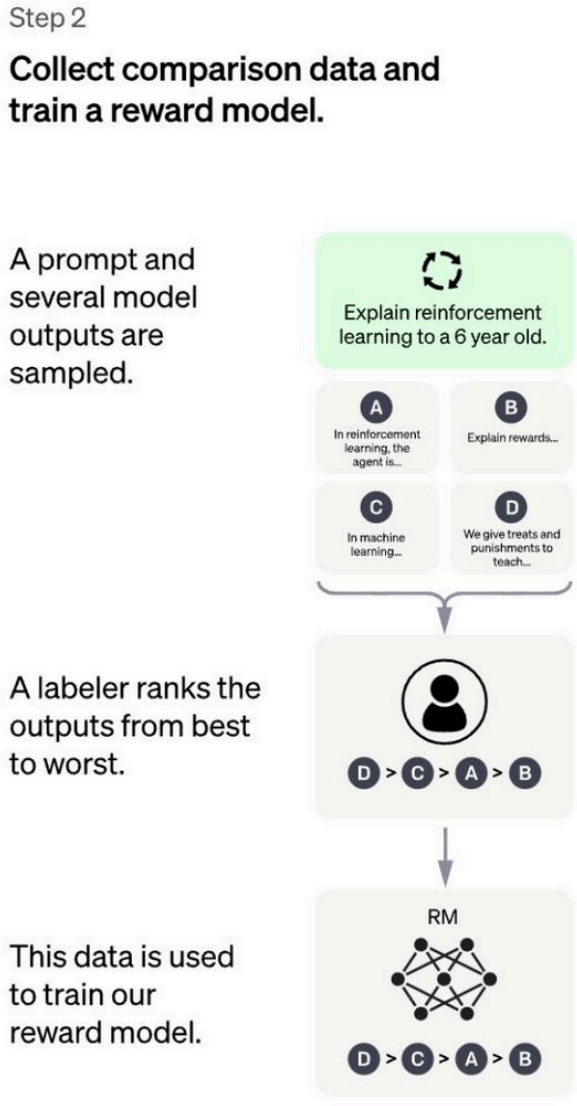
- **Objective**: Refine the foundational language model towards conversational understanding and response generation.
- **Process**:
  - Utilizes a curated dataset comprising diverse conversational prompts and corresponding human-written responses.
  - The model is fine-tuned to predict these responses accurately, aligning its outputs more closely with human-like conversational patterns.
- **Example**:
  - Prompt: "What's your favorite book and why?"
  - Model learns to generate engaging and contextually relevant responses.



Supervised Fine-Tuning Phase

Reward Modeling (RM) in ChatGPT

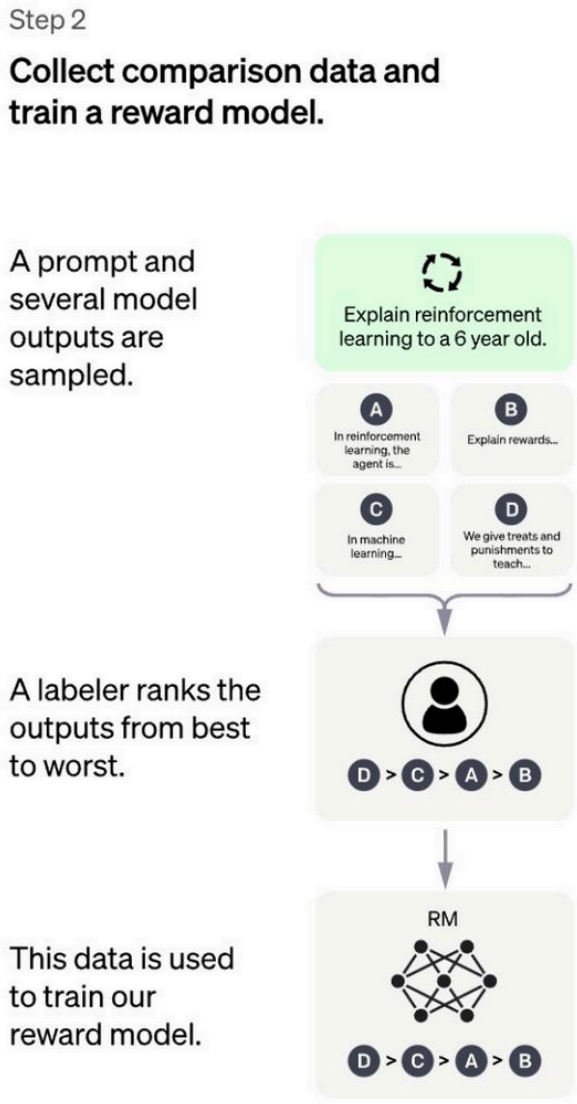
- **Objective**: Create a model to evaluate and score generated texts based on human preferences.



Reward Modeling Phase

Reward Modeling (RM) in ChatGPT

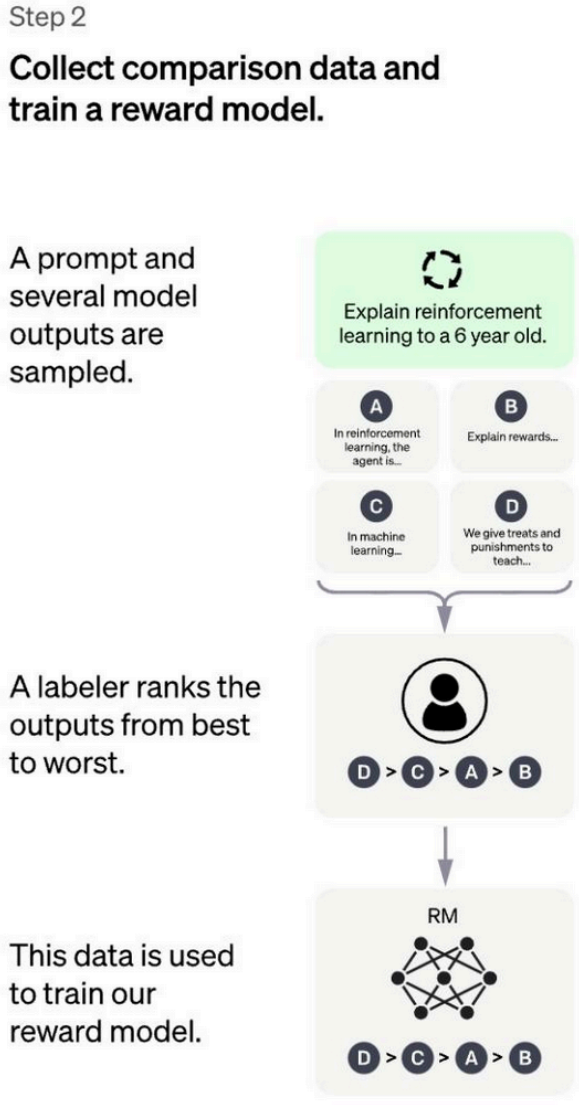
- **Objective**: Create a model to evaluate and score generated texts based on human preferences.
- **Process**:



Reward Modeling Phase

Reward Modeling (RM) in ChatGPT

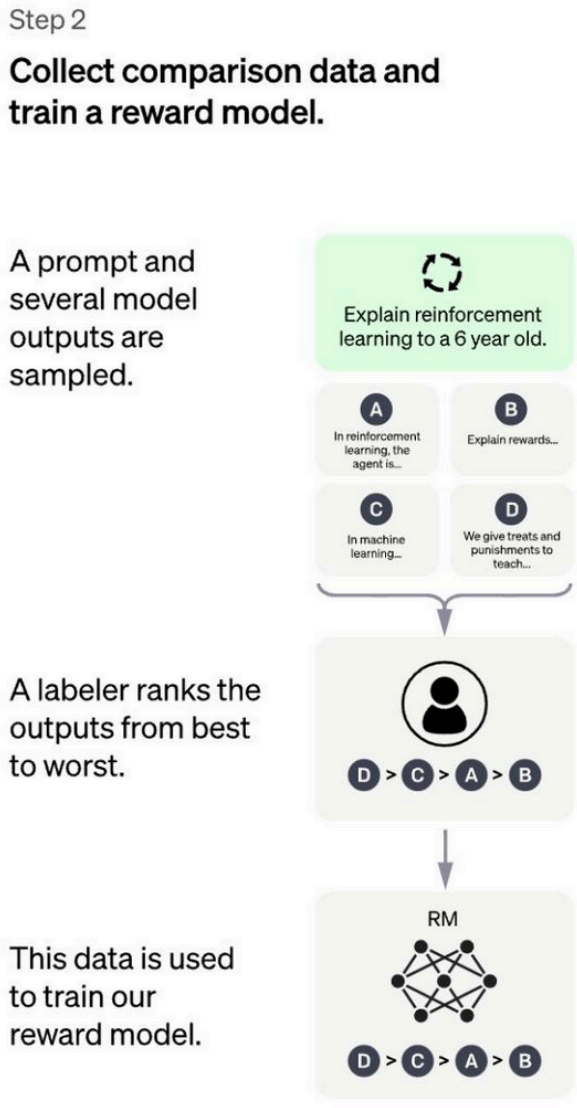
- **Objective**: Create a model to evaluate and score generated texts based on human preferences.
- **Process**:
  - Human trainers rate the quality of responses generated by the model, considering factors like relevance, coherence, and safety.



Reward Modeling Phase

Reward Modeling (RM) in ChatGPT

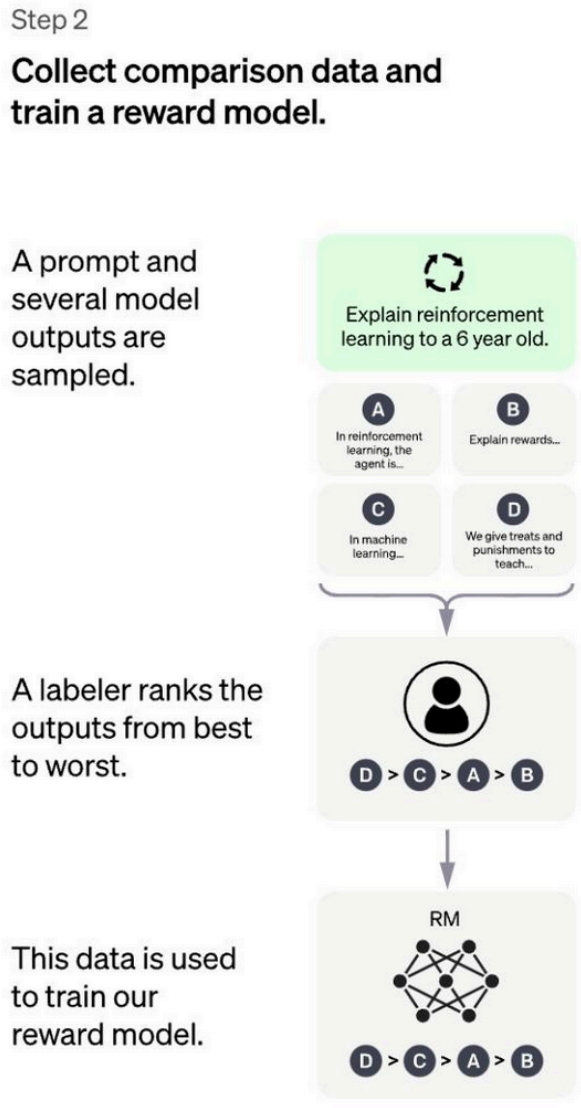
- **Objective**: Create a model to evaluate and score generated texts based on human preferences.
- **Process**:
  - Human trainers rate the quality of responses generated by the model, considering factors like relevance, coherence, and safety.
  - Use the ratings to train a separate reward model that learns to predict scores.



Reward Modeling Phase

Reward Modeling (RM) in ChatGPT

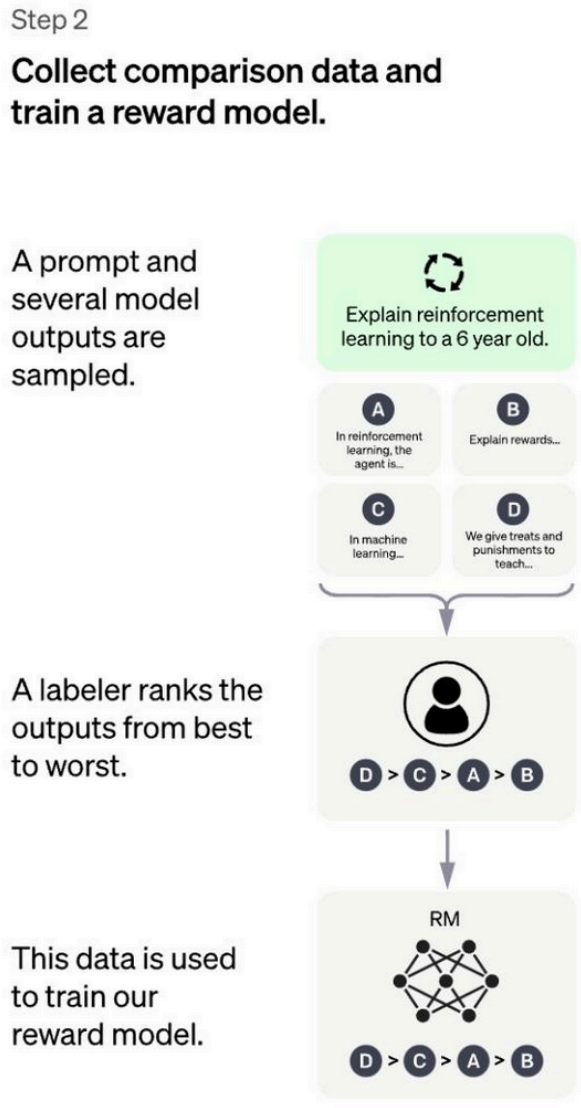
- **Objective**: Create a model to evaluate and score generated texts based on human preferences.
- **Process**:
  - Human trainers rate the quality of responses generated by the model, considering factors like relevance, coherence, and safety.
  - Use the ratings to train a separate reward model that learns to predict scores.
- **Example**:



Reward Modeling Phase

Reward Modeling (RM) in ChatGPT

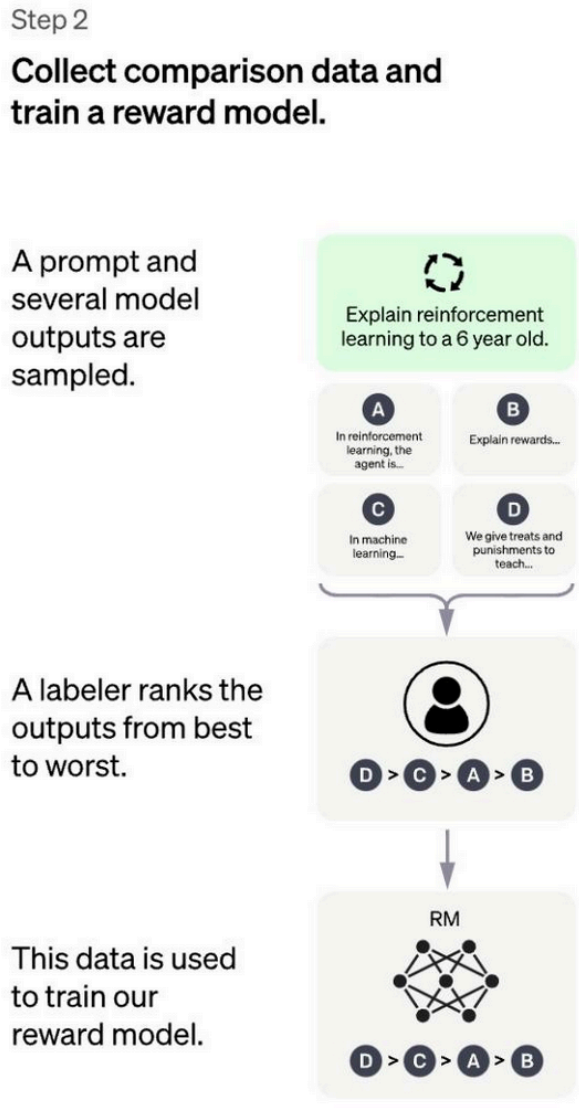
- **Objective**: Create a model to evaluate and score generated texts based on human preferences.
- **Process**:
  - Human trainers rate the quality of responses generated by the model, considering factors like relevance, coherence, and safety.
  - Use the ratings to train a separate reward model that learns to predict scores.
- **Example**:
  - Response: "My favorite book is 'To Kill a Mockingbird' because it tackles complex themes with compelling storytelling."



Reward Modeling Phase

Reward Modeling (RM) in ChatGPT

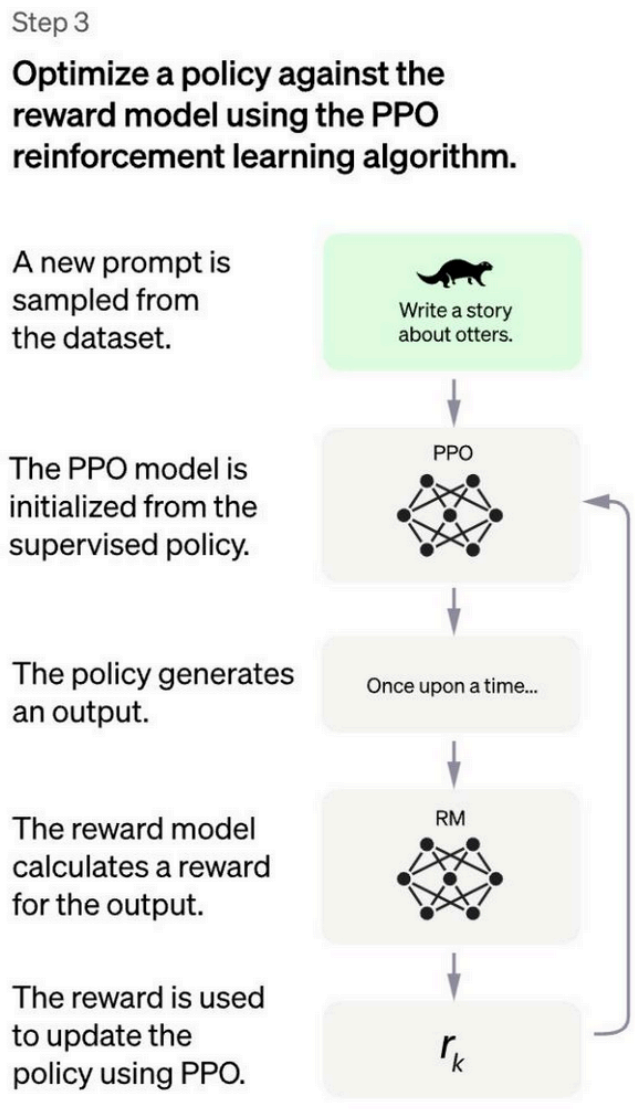
- **Objective**: Create a model to evaluate and score generated texts based on human preferences.
- **Process**:
  - Human trainers rate the quality of responses generated by the model, considering factors like relevance, coherence, and safety.
  - Use the ratings to train a separate reward model that learns to predict scores.
- **Example**:
  - Response: "My favorite book is 'To Kill a Mockingbird' because it tackles complex themes with compelling storytelling."
  - Reward model learns to score such responses for effectiveness and relevance.



Reward Modeling Phase

Proximal Policy Optimization (PPO) in ChatGPT

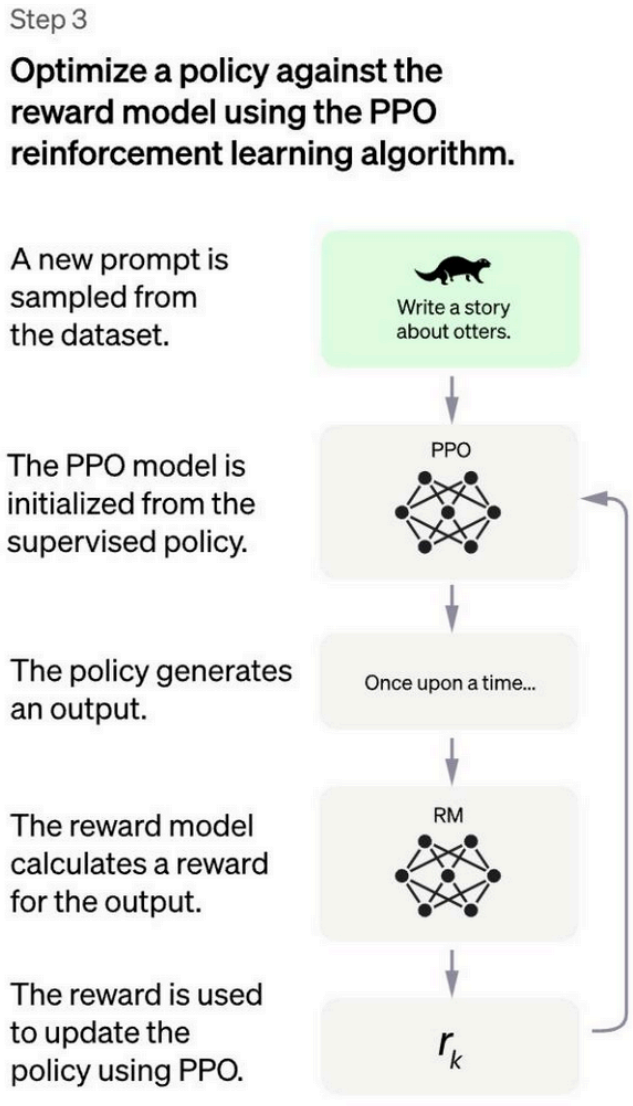
- **Objective**: Refine ChatGPT's responses to align with human preferences.



PPO Phase

Proximal Policy Optimization (PPO) in ChatGPT

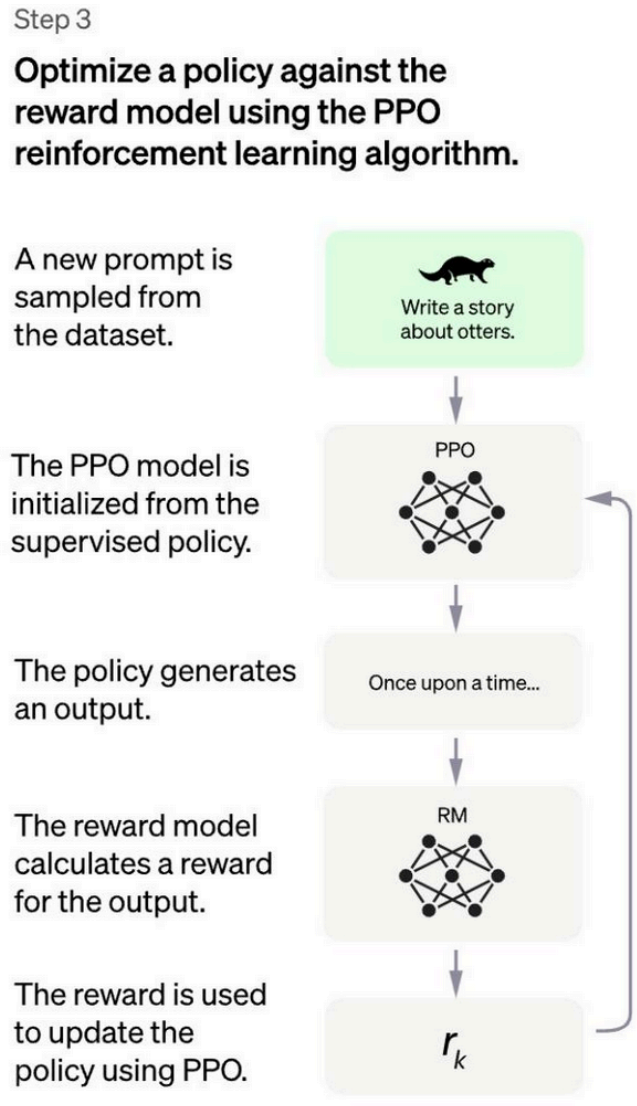
- **Objective**: Refine ChatGPT's responses to align with human preferences.
- **Process**:



PPO Phase

Proximal Policy Optimization (PPO) in ChatGPT

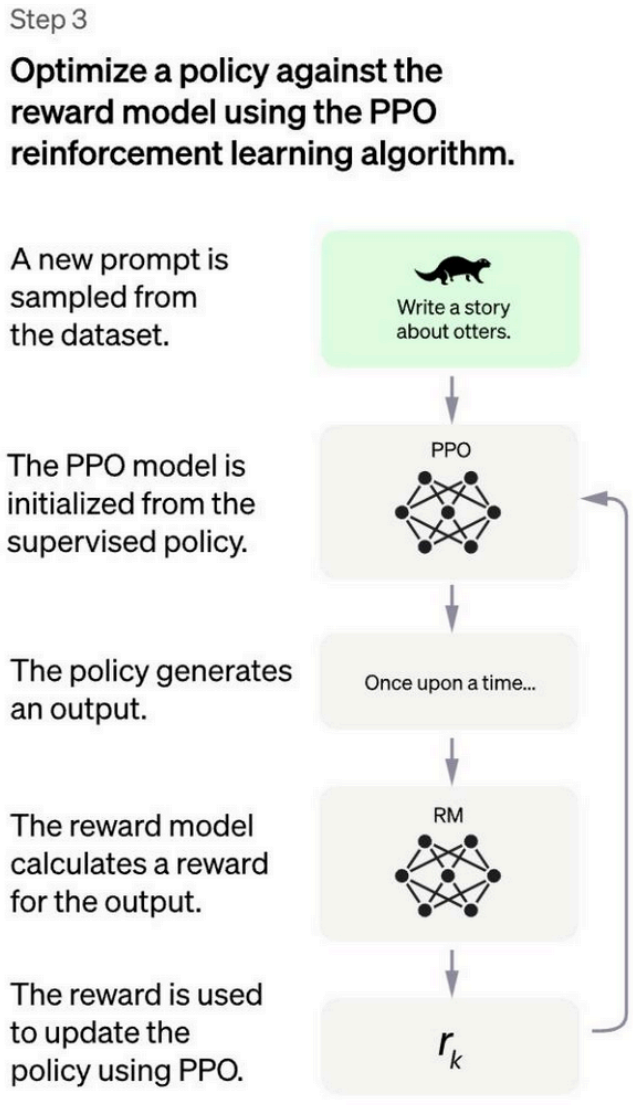
- **Objective**: Refine ChatGPT's responses to align with human preferences.
- **Process**:
  - The model's training is guided by the reward model to generate responses that are likely to be scored highly.



PPO Phase

Proximal Policy Optimization (PPO) in ChatGPT

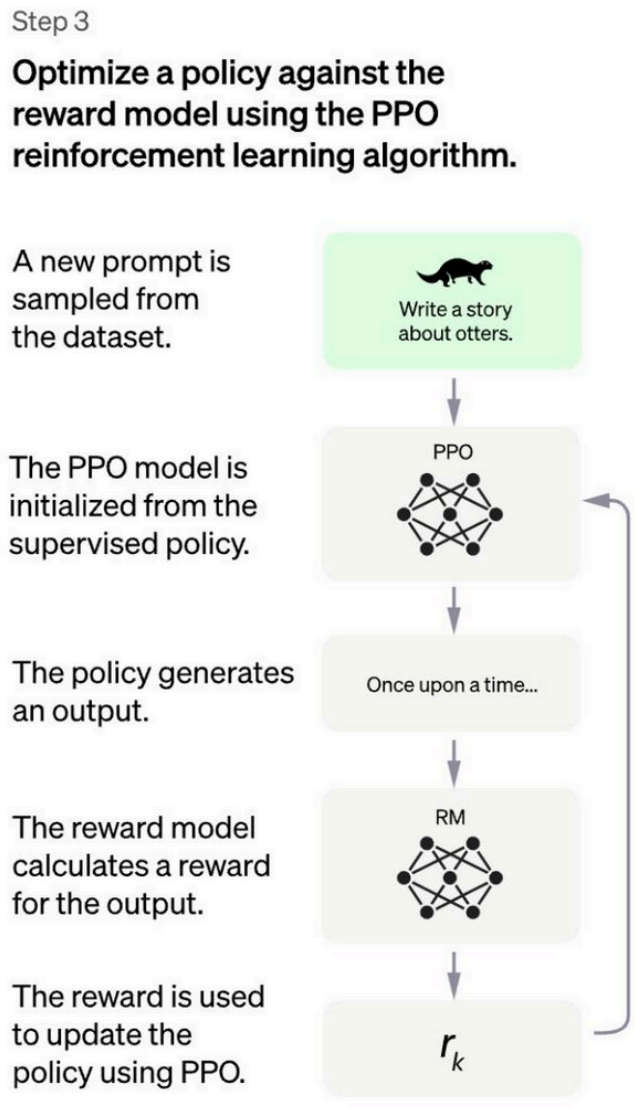
- **Objective**: Refine ChatGPT's responses to align with human preferences.
- **Process**:
  - The model's training is guided by the reward model to generate responses that are likely to be scored highly.
  - Iteratively adjusts the model's parameters to maximize the expected reward from the reward model.



PPO Phase

Proximal Policy Optimization (PPO) in ChatGPT

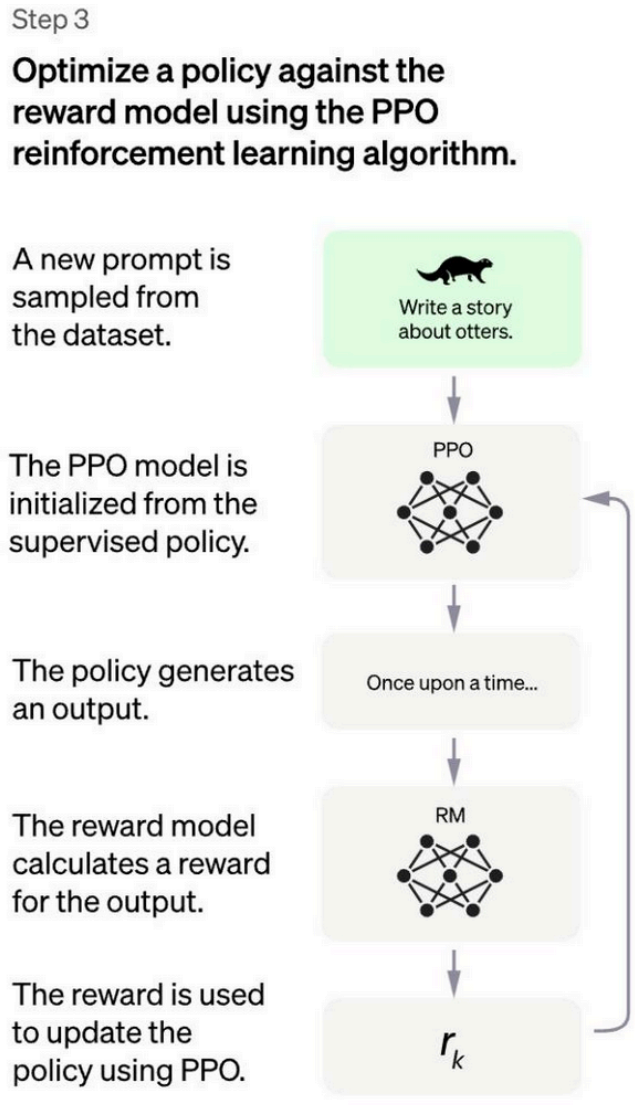
- **Objective**: Refine ChatGPT's responses to align with human preferences.
- **Process**:
  - The model's training is guided by the reward model to generate responses that are likely to be scored highly.
  - Iteratively adjusts the model's parameters to maximize the expected reward from the reward model.
- **Example**:



PPO Phase

Proximal Policy Optimization (PPO) in ChatGPT

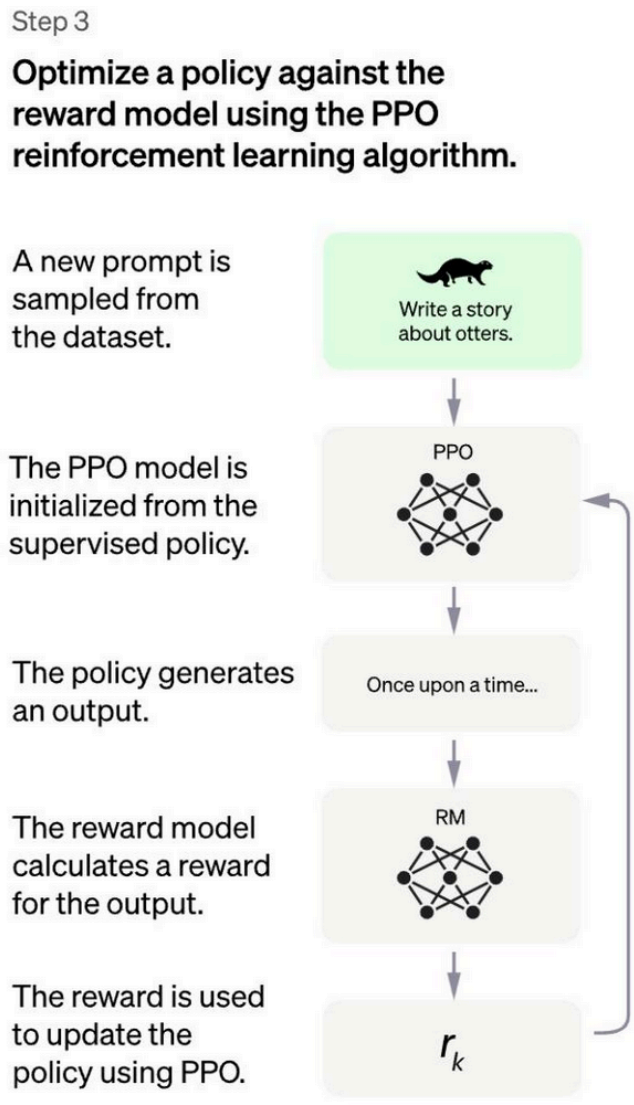
- **Objective**: Refine ChatGPT's responses to align with human preferences.
- **Process**:
  - The model's training is guided by the reward model to generate responses that are likely to be scored highly.
  - Iteratively adjusts the model's parameters to maximize the expected reward from the reward model.
- **Example**:
  - The model generates a variety of responses to a prompt.



PPO Phase

Proximal Policy Optimization (PPO) in ChatGPT

- **Objective**: Refine ChatGPT's responses to align with human preferences.
- **Process**:
  - The model's training is guided by the reward model to generate responses that are likely to be scored highly.
  - Iteratively adjusts the model's parameters to maximize the expected reward from the reward model.
- **Example**:
  - The model generates a variety of responses to a prompt.
  - It then estimates the reward for each response and prefers choices that maximize this reward, leading to more human-aligned responses.



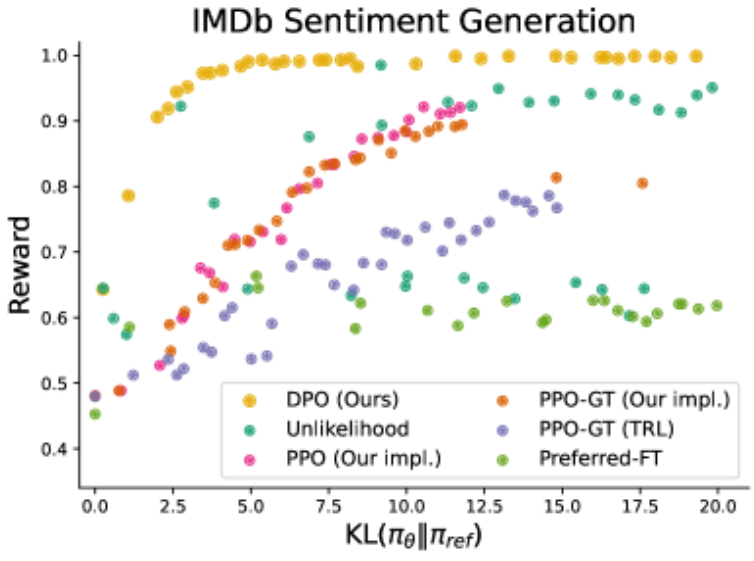
PPO Phase

Direct Preference Optimization (DPO)

Problem with PPO: 3 models in memory (policy, reward, reference), an unstable RL loop, a reward model that can be gamed.

DPO's insight (Rafailov et al., 2023):

- The "reward" of an answer is just the **log-ratio** between policy and a **frozen reference**: $\log \frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)}$.
- A **preference** $(y_w \succ y_l)$ is then just **comparing those two log-ratios**: training becomes a **binary classification** (prefer y_w or y_l?), with no reward model and no RL loop.



DPO matches/beats PPO on the reward–KL frontier. Rafailov et al. (2023)

Gains:

- Preference data is easier to collect than reward labels.
- Comparable to or better than PPO, and far more stable (plot).

Beyond DPO: the Preference-Optimization Zoo

- **DPO** (Rafailov et al., 2023): closed-form, stable, the de-facto baseline.

Beyond DPO: the Preference-Optimization Zoo

- **DPO** (Rafailov et al., 2023): closed-form, stable, the de-facto baseline.
- **IPO**: adds regularization to avoid overfitting the preference pairs.
- **ORPO**: merges SFT and preference optimization in a **single stage**, no reference model.
- **SimPO**: reference-free, length-normalized reward $\Rightarrow$ less verbosity bias, lower memory.
- **KTO**: learns from **individual** good/bad labels, no need for paired data.

Beyond DPO: the Preference-Optimization Zoo

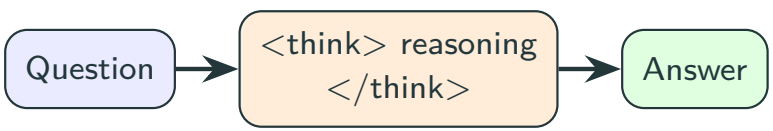
- **DPO** (Rafailov et al., 2023): closed-form, stable, the de-facto baseline.
- **IPO**: adds regularization to avoid overfitting the preference pairs.
- **ORPO**: merges SFT and preference optimization in a **single stage**, no reference model.
- **SimPO**: reference-free, length-normalized reward $\Rightarrow$ less verbosity bias, lower memory.
- **KTO**: learns from **individual** good/bad labels, no need for paired data.
- **RLAIF**: replace human labels with an LLM judge (AI feedback) to scale data collection.
- **GRPO** (DeepSeek, 2024): group-relative RL, no value network, now standard for **reasoning** models.

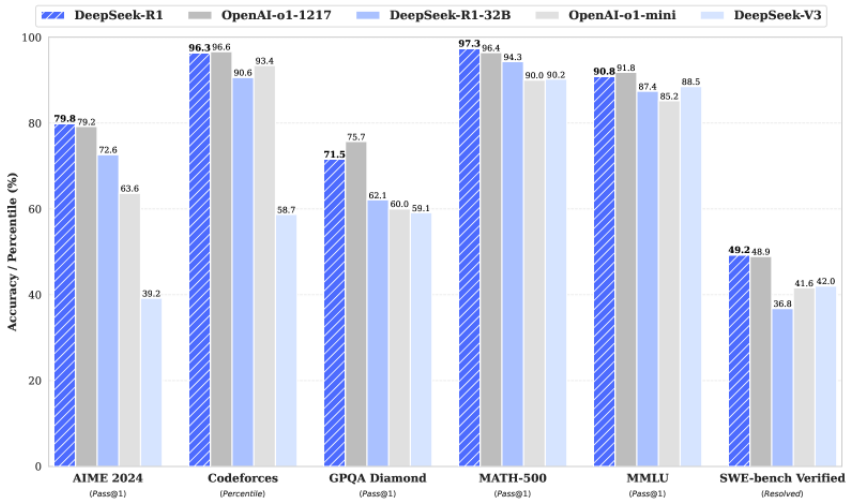
See Mantis NLP / Argilla / Hugging Face TRL blog posts for hands-on recipes.

Reasoning Models: Thinking Before Answering

o1, o3, DeepSeek-R1, Gemini 2.5, Claude (2024–2025): models trained to emit a long internal **chain of thought** before the final answer.

- **Link to CoT:** prompt-time "think step by step" is now **learned** and baked into the weights, not asked for.
- Spend more **test-time compute** on hard problems (math, code, science): longer trace = better answer.
- Trade-off: stronger reasoning, but higher latency and token cost.





DeepSeek-R1 vs o1 (Pass@1).

DeepSeek-AI (2025)

How Reasoning Models Are Trained: The Core Problem

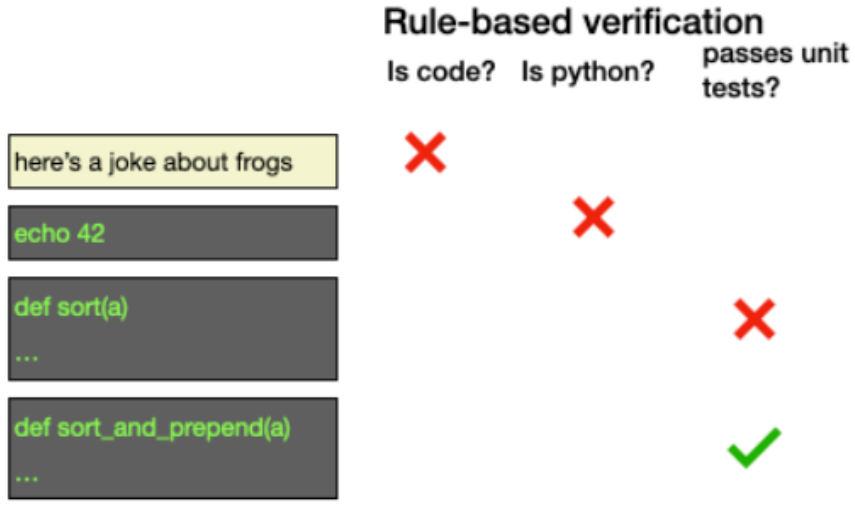
The bottleneck: teaching reasoning by SFT needs **high-quality reasoning traces**, and writing them **by hand does not scale**.

The key idea, RLVR (RL on *verifiable* rewards):

- No reward model for math/code: the answer is **checkable** (math: exact match, code: unit tests).
- The model **generates its own reasoning**, rewarded only when the answer is right.

DeepSeek-R1-Zero: pure RL (GRPO) on the base model. A strong **general reasoning model emerges** (self-correction, "aha moments").

Limitation: outputs are **unreadable**, a solver, not an assistant.



Rule based verification for

"Write a python code to sort a list of number"

Credit: Jay Alammar

From R1-Zero to R1: Cold Start & Data Factory

Fix: train a new "R1-Zero", but with a **cold start** that frames the outputs into a clean, readable format.

1. **Base:** DeepSeek-V3-Base (pre-trained).
2. **Intermediate model:** cold-start SFT on ∼thousands of human CoT examples, then the same reasoning RL → now clean and readable. This becomes a **data factory**.
3. **Generate data** (rejection sampling from the intermediate model): ∼600k reasoning + ∼200k non-reasoning = ∼800k high-quality samples.
4. **Final R1:** SFT the ∼800k on V3-Base, then a **final RL** stage (verifiable rewards + general reward model + safety).

Distillation: SFT those ∼800k samples into smaller models (Qwen, Llama) ⇒ cheap reasoning at small scale, with no RL needed.

Training of ChatGPT: Process and Innovations

- **Differences from Previous Models**:

Training of ChatGPT: Process and Innovations

- **Differences from Previous Models**:
  - *Interactive Feedback*: Incorporation of dialogues and human interaction nuances.

Training of ChatGPT: Process and Innovations

- **Differences from Previous Models**:
  - *Interactive Feedback*: Incorporation of dialogues and human interaction nuances.
  - *Dynamic Learning*: Ability to learn from user interactions and adapt responses.

Training of ChatGPT: Process and Innovations

- **Differences from Previous Models**:
  - *Interactive Feedback*: Incorporation of dialogues and human interaction nuances.
  - *Dynamic Learning*: Ability to learn from user interactions and adapt responses.
  - *Ethical and Safety Considerations*: Enhanced focus on generating safe, ethical, and contextually appropriate responses.

Applications & other models

Cursor: AI-Powered Code Editor

- **What it is**: A code editor with built-in AI assistance to enhance coding productivity.

Cursor: AI-Powered Code Editor

- **What it is**: A code editor with built-in AI assistance to enhance coding productivity.
- **How it works**:

Cursor: AI-Powered Code Editor

- **What it is**: A code editor with built-in AI assistance to enhance coding productivity.
- **How it works**:
  - Integrates a powerful LLM for code completions, refactoring, and natural language explanations.

Cursor: AI-Powered Code Editor

- **What it is**: A code editor with built-in AI assistance to enhance coding productivity.
- **How it works**:
  - Integrates a powerful LLM for code completions, refactoring, and natural language explanations.
  - Supports seamless code navigation and AI-enhanced coding workflows.

Cursor: AI-Powered Code Editor

- **What it is**: A code editor with built-in AI assistance to enhance coding productivity.
- **How it works**:
  - Integrates a powerful LLM for code completions, refactoring, and natural language explanations.
  - Supports seamless code navigation and AI-enhanced coding workflows.
  - Provides a chat with agent or manual inputs directly within the editor.

Cursor: AI-Powered Code Editor

- **What it is**: A code editor with built-in AI assistance to enhance coding productivity.
- **How it works**:
  - Integrates a powerful LLM for code completions, refactoring, and natural language explanations.
  - Supports seamless code navigation and AI-enhanced coding workflows.
  - Provides a chat with agent or manual inputs directly within the editor.
- **ROI**:

Cursor: AI-Powered Code Editor

- **What it is**: A code editor with built-in AI assistance to enhance coding productivity.
- **How it works**:
  - Integrates a powerful LLM for code completions, refactoring, and natural language explanations.
  - Supports seamless code navigation and AI-enhanced coding workflows.
  - Provides a chat with agent or manual inputs directly within the editor.
- **ROI**:
  - Speeds up coding tasks with smart AI-based code generation.

Cursor: AI-Powered Code Editor

- **What it is**: A code editor with built-in AI assistance to enhance coding productivity.
- **How it works**:
  - Integrates a powerful LLM for code completions, refactoring, and natural language explanations.
  - Supports seamless code navigation and AI-enhanced coding workflows.
  - Provides a chat with agent or manual inputs directly within the editor.
- **ROI**:
  - Speeds up coding tasks with smart AI-based code generation.
  - Enhances the development process through quick code insights and debugging support.

LlamaIndex: Leveraging RAG for Internal Data

- **What it is**: A system that utilizes Retrieval-Augmented Generation (RAG) with LLMs for internal data querying and analysis.

LlamaIndex

LlamaIndex: Leveraging RAG for Internal Data

- **What it is**: A system that utilizes Retrieval-Augmented Generation (RAG) with LLMs for internal data querying and analysis.
- **How it works**:

LlamaIndex

LlamaIndex: Leveraging RAG for Internal Data

- **What it is**: A system that utilizes Retrieval-Augmented Generation (RAG) with LLMs for internal data querying and analysis.
- **How it works**:
  - Combines the power of LLMs with a retrieval system to fetch relevant documents.

LlamaIndex

LlamaIndex: Leveraging RAG for Internal Data

- **What it is**: A system that utilizes Retrieval-Augmented Generation (RAG) with LLMs for internal data querying and analysis.
- **How it works**:
  - Combines the power of LLMs with a retrieval system to fetch relevant documents.
  - Enhances the generation of responses by conditioning on retrieved documents.

LlamaIndex

LlamaIndex: Leveraging RAG for Internal Data

- **What it is**: A system that utilizes Retrieval-Augmented Generation (RAG) with LLMs for internal data querying and analysis.
- **How it works**:
  - Combines the power of LLMs with a retrieval system to fetch relevant documents.
  - Enhances the generation of responses by conditioning on retrieved documents.
- **Usage**:

LlamaIndex

LlamaIndex: Leveraging RAG for Internal Data

- **What it is**: A system that utilizes Retrieval-Augmented Generation (RAG) with LLMs for internal data querying and analysis.
- **How it works**:
  - Combines the power of LLMs with a retrieval system to fetch relevant documents.
  - Enhances the generation of responses by conditioning on retrieved documents.
- **Usage**:
  - Facilitates complex queries on internal datasets.

LlamaIndex

LlamaIndex: Leveraging RAG for Internal Data

- **What it is**: A system that utilizes Retrieval-Augmented Generation (RAG) with LLMs for internal data querying and analysis.
- **How it works**:
  - Combines the power of LLMs with a retrieval system to fetch relevant documents.
  - Enhances the generation of responses by conditioning on retrieved documents.
- **Usage**:
  - Facilitates complex queries on internal datasets.
  - Provides contextually enriched answers by combining generative power with specific data retrieval.

LlamaIndex

Vera: Your Trusted Number for Fact-Checking

Functionality:

- Vera is a single, free-to-use app for verifying facts and combating misinformation.
- Provides users with quick and reliable answers to fact-check claims.

Benefits:

- **Accessibility:** A simple and direct solution to access fact-checked information.
- **Public Trust:** Promotes transparency and helps build trust in information sources.
- **Countering Misinformation:** Acts as a frontline tool in the fight against misinformation and polarization.

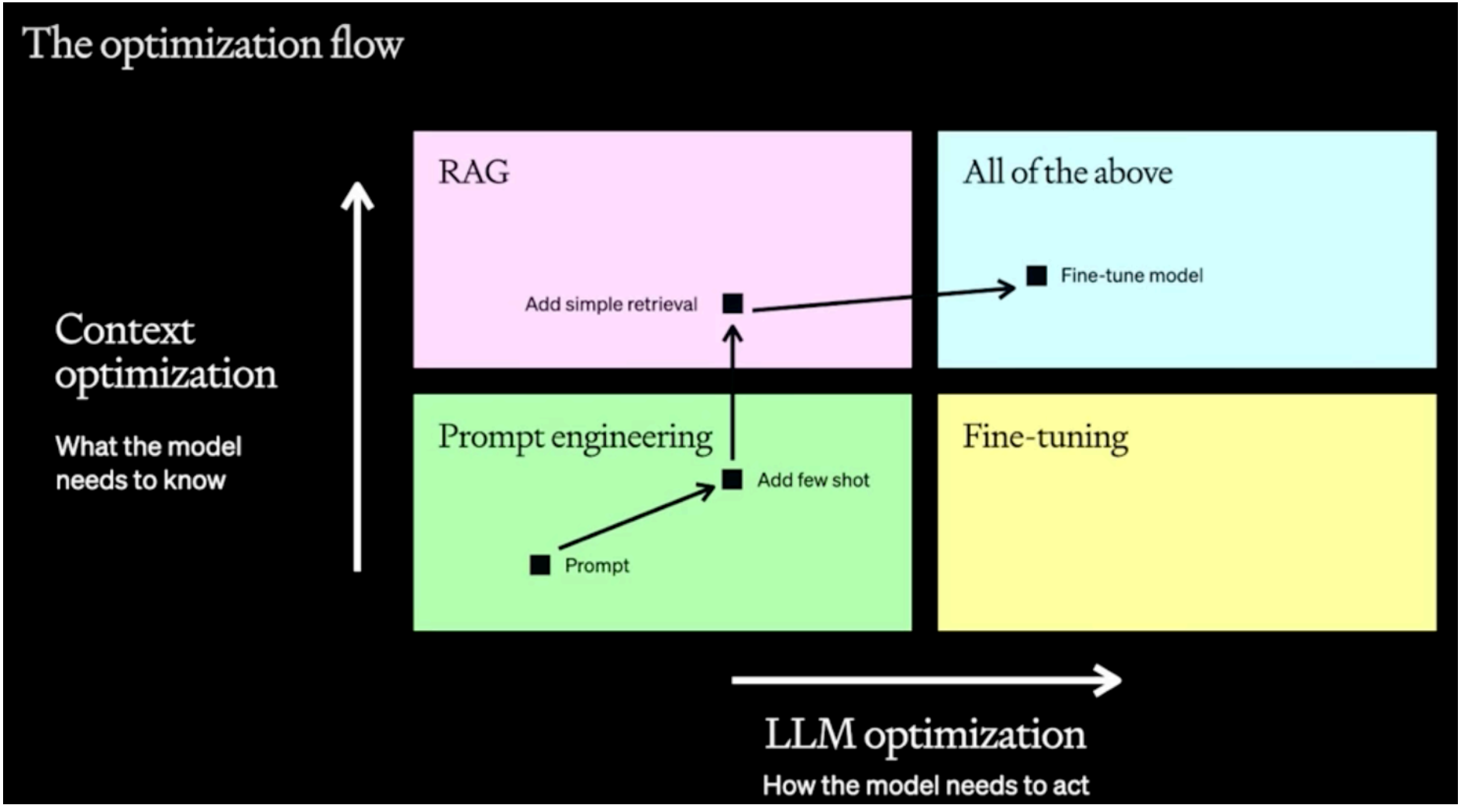
Vera: `askvera.org`

Maximizing LLM Performance

Techniques to Utilize LLMs

- **Prompt Engineering:** Crafting prompts that guide the LLM to generate the desired output. Asking the model to act in a specific way, leveraging pre-trained knowledge to fulfill complex tasks.
- **Retrieval-Augmented Generation (RAG):** Optimizing the context by providing the model with external knowledge to know before generating a response. This method enhances the model's ability to generate more contextually relevant answers.
- **Fine-Tuning:** Training the LLM on a specific dataset to optimize its performance for a particular task. It's about how the model needs to act, refining its responses based on additional training to align with the task's requirements.
- **Combining Techniques:** While each technique has its strengths, combining prompt engineering, RAG, and fine-tuning can offer a comprehensive approach to leveraging LLMs. The best direction depends on the specific use case.

Techniques to Utilize LLMs



From openAI demo day

Prompt Engineering with LLMs

Techniques for Enhanced Interaction:

- **LLM-Enhanced Prompts:** Utilizing LLMs to refine prompts for better accuracy and relevance. eg. Liu et al., 2023 - "Dynamic LLM-Agent Network"
- **Few-Shot Learning:** Incorporating examples within prompts to guide LLMs towards desired outputs. *Reference: GPT-3, OpenAI.*
- **Chain of Thoughts:** Encouraging LLMs to "think aloud," enhancing reasoning for complex queries. eg. Wei et al., 2022 - "Chain of Thought Prompting Elicits Reasoning in Large Language Models."
- **Schema-Constrained Output:** Structuring prompts to yield outputs in specific formats, like JSON for NER tasks. eg. Shin et al., 2021 - "Constrained Language Models Yield Few-Shot Semantic Parsers."

Using LLMs for Prompt Optimization

Python Example with OpenAI API:

Code Snippet

```
from openai import OpenAI
client = OpenAI(api_key='your-api-key-here')

prompt = """
I'd like to understand the two main themes of the movie description.
Please provide a list of two themes of it:

{{MOVIE}}
"""

messages = [{"role": "system",
             "content": "You're a prompt engineer that needs to
                        optimize a prompt. I'll give you a prompt
                        and you will and objective and
                        you'll improve the prompt."},
            {"role": "user", "content": prompt}]
```

Using LLMs for Prompt Optimization

Python Example with OpenAI API:

Code Snippet

```
response = client.chat.completions.create(
  model="gpt-4o",
  messages=messages,
  temperature=0.7,
  max_tokens=60,
  top_p=1.0,
  frequency_penalty=0.0,
  presence_penalty=0.0
)

print(response.choices[0].message.content.strip())
```

Understanding Temperature in LLMs

Temperature in Language Models:

- Controls the randomness in the prediction of the next word.
- A *lower temperature* (e.g., 0.1) results in more deterministic and confident outputs, often repeating the most likely words.
- A *higher temperature* (e.g., 1.0 or higher) increases diversity in generated text, producing more varied and sometimes more creative or unexpected results.

Understanding Temperature in LLMs

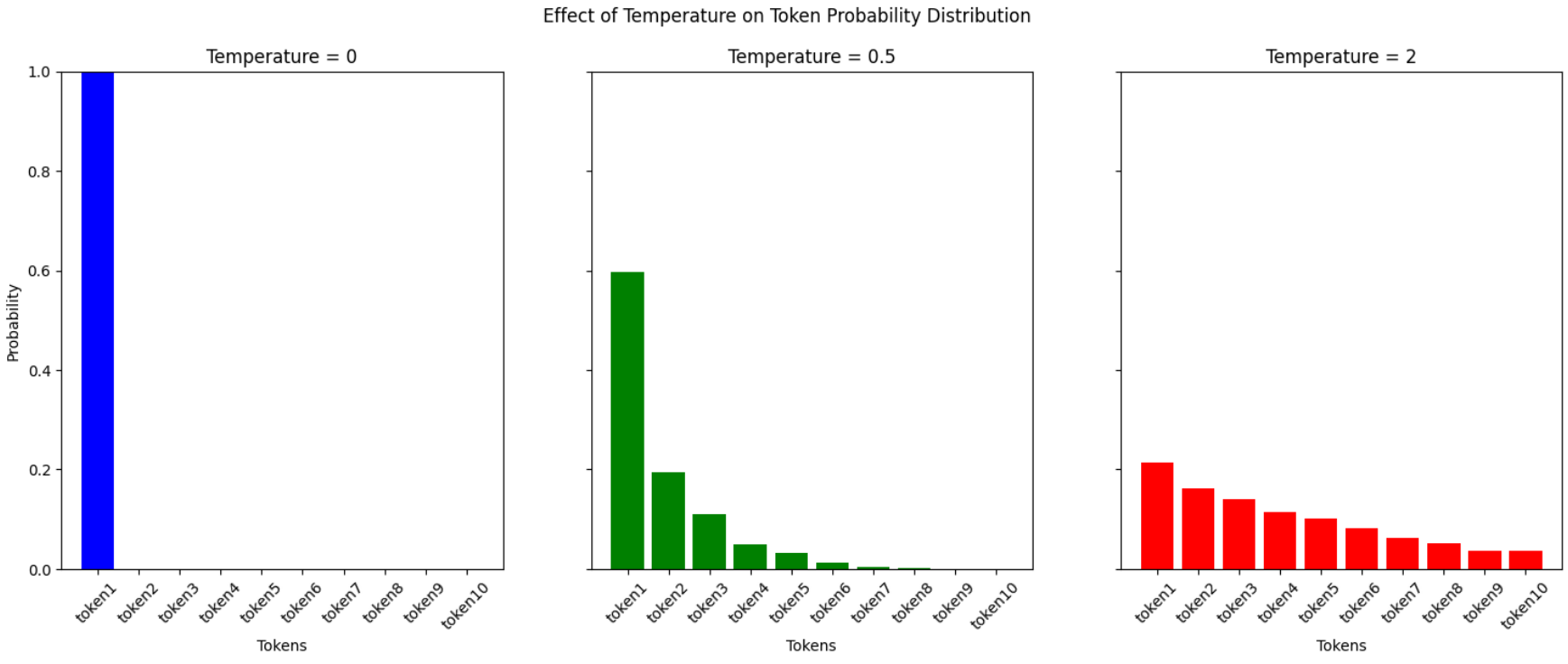
Temperature in Language Models:

- Controls the randomness in the prediction of the next word.
- A *lower temperature* (e.g., 0.1) results in more deterministic and confident outputs, often repeating the most likely words.
- A *higher temperature* (e.g., 1.0 or higher) increases diversity in generated text, producing more varied and sometimes more creative or unexpected results.

Temperature Effect: Prompt: "The sun sets over the"

- *Low Temperature (0.1)*: "The sun sets over the horizon."
- *Medium Temperature (0.7)*: "The sun sets over the distant mountains, casting a golden hue."
- *High Temperature (1.0)*: "The sun sets over the sea, weaving tales of ancient mariners and distant shores."

Understanding Temperature in LLMs



Higher temperature, smoother distribution

Understanding Top-p Sampling in LLMs

Top-p Sampling in Language Models:

- Selects the smallest set of words whose cumulative probability exceeds a threshold p.
- It dynamically adjusts the size of the considered vocabulary based on the p value, focusing on a more probable subset for each prediction.
- This approach helps balance between creativity and relevance by avoiding the less probable, and hence more random, words without being overly deterministic.

Understanding Top-p Sampling in LLMs

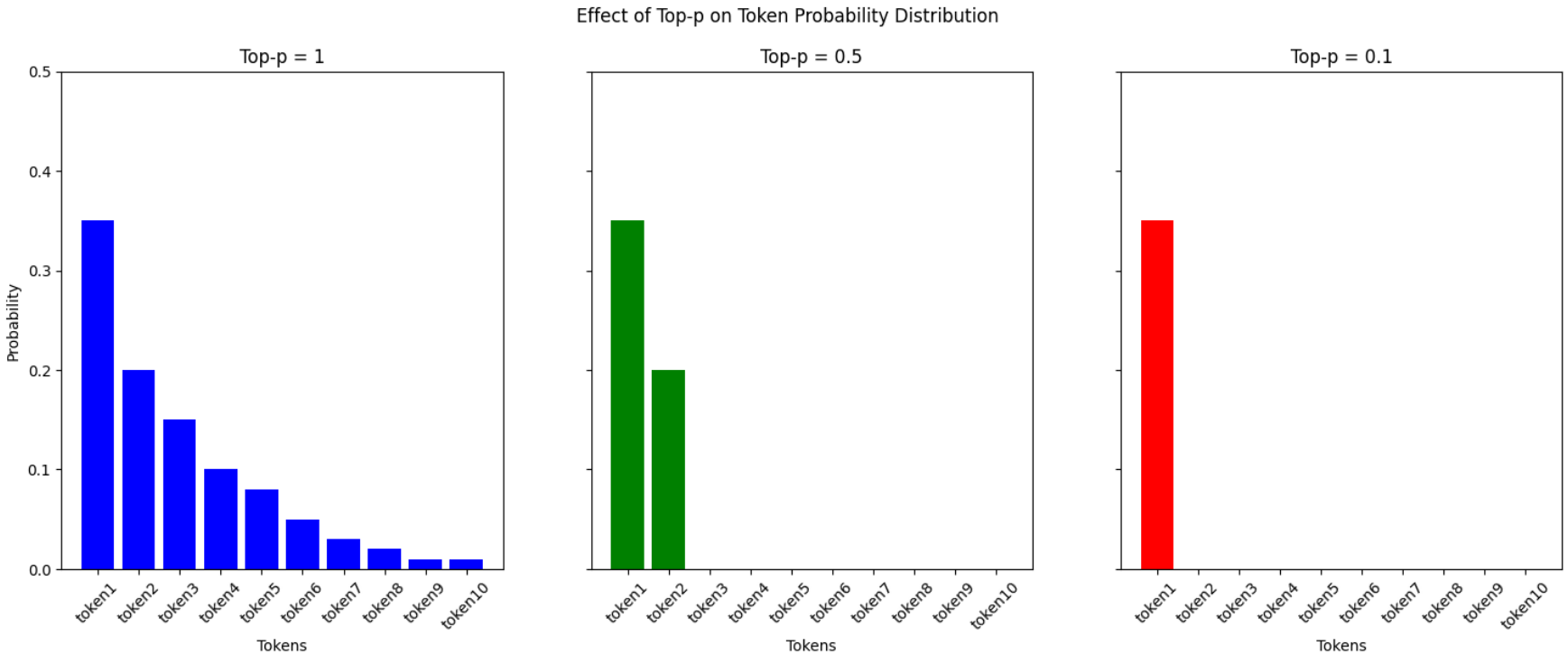
Top-p Sampling in Language Models:

- Selects the smallest set of words whose cumulative probability exceeds a threshold p.
- It dynamically adjusts the size of the considered vocabulary based on the p value, focusing on a more probable subset for each prediction.
- This approach helps balance between creativity and relevance by avoiding the less probable, and hence more random, words without being overly deterministic.

Top-p Sampling Effect: Prompt: "In the distant future, humanity"

- *Low p Value (0.2)*: "survives in a utopia."
- *Medium p Value (0.5)*: "explores new galaxies, seeking life."
- *High p Value (0.9)*: "faces challenges beyond imagination, like AI revolutions and interstellar wars."

Understanding Top-p Sampling in LLMs



Lower top_p, smaller tokens set to consider

Temperature vs. Top-p: Which One Should You Tune?

Rule of thumb: tune *one*, not both. OpenAI/Anthropic docs explicitly recommend it: their interaction is hard to reason about and hurts reproducibility.

- **Start with temperature.** It is the most intuitive dial and changes *diversity*, not the model's underlying reasoning.
  - Correctness tasks (extraction, classification, code, tool use): **T ≈ 0–0.3** and validate output structurally.
  - Creative / brainstorming tasks: T ≈ 0.7–1.0.
- **Top-p** mainly caps the long tail; useful at higher temperature to avoid incoherent tokens. Leave at 1.0 if you already tune T.

Note: at T = 0 the output is (near-)deterministic, so top-p is irrelevant. Recent work (Min-p, ICLR 2025) shows tail-cutting matters most at *high* temperature.

Few-Shot Learning Techniques for LLM Prompting

FSL techniques empower LLMs to adapt and generalize enhancing their ability to comprehend and respond to diverse prompts.

Main Advantages:

- **Flexibility**: LLMs can learn from a small number of examples, enabling adaptation to various tasks and contexts.
- **Efficiency**: Requires minimal labeled data, reducing the annotation burden and facilitating rapid model customization.
- **Generalization**: Promotes robustness and adaptability by extracting common patterns and concepts from limited examples.

Why Few-Shot Learning Works Better:

LLMs' extensive pre-training allows them to leverage prior knowledge and patterns from diverse domains, enabling effective transfer learning with few-shot examples.

Implementing Few-Shot Learning with OpenAI API

Python Example with OpenAI API:

Python Code Example

```
from openai import OpenAI
client = OpenAI(api_key="your-api-key-here")

few_shot_examples = ["My name is Harry\n---\nJe m'appelle Harry.",
                     "I love pizzas\n---\nJ'adore les pizzas."]

messages = [{"role": "system",
             "content": "You are an English to French translator."},
            {"role": "user",
             "content": "Here are some examples:\n" +
                        "\n\n".join(few_shot_examples)},
            {"role": "user",
             "content": "Translate to French:\nHello world"}]
```

Implementing Few-Shot Learning with OpenAI API

Python Code Example

```
# Perform few-shot learning with OpenAI API
response = client.chat.completions.create(
    model="gpt-4o",
    messages=messages,
    temperature=0.7,
    max_tokens=100
)

# Print the generated response
print(response.choices[0].message.content.strip())
```

Chain of Thoughts in LLMs

Chain of Thoughts is a technique used to guide the generation of responses in Large Language Models (LLMs) by breaking down the thought process into smaller, sequential steps.

- **Sequential Generation:** LLMs are prompted with a series of interconnected thoughts or questions, each building upon the previous one.
- **Structured Outputs:** By structuring the input as a chain of related thoughts, LLMs are encouraged to produce coherent and contextually relevant responses.
- **Enhanced Understanding:** This technique helps LLMs understand the context and intent better, leading to more accurate and meaningful outputs.
- **Improved Communication:** CoT facilitates more natural and engaging conversations, mimicking human thought processes.

Implementing Chain of Thoughts with OpenAI API

Python Example with OpenAI API:

Python Code Example

```
prompt = """
Q: What is the value of 5!?
A: 5! = 1 x 2 x 3 x 4 x 5, so step by step:
   1 x 2 = 2, 2 x 3 = 6, 6 x 4 = 24, 24 x 5 = 120.
   The answer is 120.

Q: What is the value of (3 x 100) + 5 - (42 / 7)?
A: Let's think step by step.
"""

# Generate response using OpenAI API
response = client.chat.completions.create(
    model="gpt-4o",
    messages=[{"role": "user", "content": prompt}],
    temperature=0.7,
    max_tokens=100)
```

Constraining LLM Outputs with JSON Schema

JSON schema provides a powerful way to define the structure of outputs from LLMs, ensuring that generated text adheres to specific formats or contains particular types of information.

- **Defining the Schema:** Specify the expected output format using JSON schema, including types of data, required fields, and descriptions.
- **Applying to LLMs:** Pass the schema in the request so the model is *constrained* to emit valid structured output.
- **Modern enforcement:** OpenAI `response_format` / Structured Outputs, function calling, or open-source constrained decoding (Outlines, vLLM) guarantee valid JSON.
- **Use case:** entity extraction (NER), forms, feeding downstream APIs.
- **Benefit:** reliable, machine-readable outputs in production.

Structured Outputs with response_format (1/2)

Define the schema as a Pydantic model:

Python Code Example

```
from openai import OpenAI
from pydantic import BaseModel

client = OpenAI(api_key="your-api-key-here")

class Person(BaseModel):
    name: str
    occupation: str
    is_student: bool

prompt = "My name is Arnault and I work as a lecturer at BSE in Barcelona."
```

Structured Outputs with response_format (2/2)

The model is forced to return valid JSON matching the schema:

Python Code Example

```
response = client.chat.completions.parse(
    model="gpt-4o",
    messages=[
        {"role": "system", "content": "Extract the person's info."},
        {"role": "user", "content": prompt},
    ],
    response_format=Person,
)

person = response.choices[0].message.parsed
# Person(name='Arnault', occupation='lecturer', is_student=False)
```

Retrieval-Augmented Generation

Retrieval-Augmented Generation Methods

1. **Naive RAG:**
   - Simple retrieval process to fetch relevant passages from a KB.
   - Retrieved information is concatenated with the query to augment the context before generation.
2. **Advanced RAG:**
   - Improve each stage: better chunking, query rewriting, hierarchical / sentence-window retrieval, and **reranking** of candidates.
3. **Modular / Agentic RAG:**
   - Multi-step retrieval, tool use, and self-correction loops (covered with agents in Session 10).

Naive RAG Principle

How It Works:

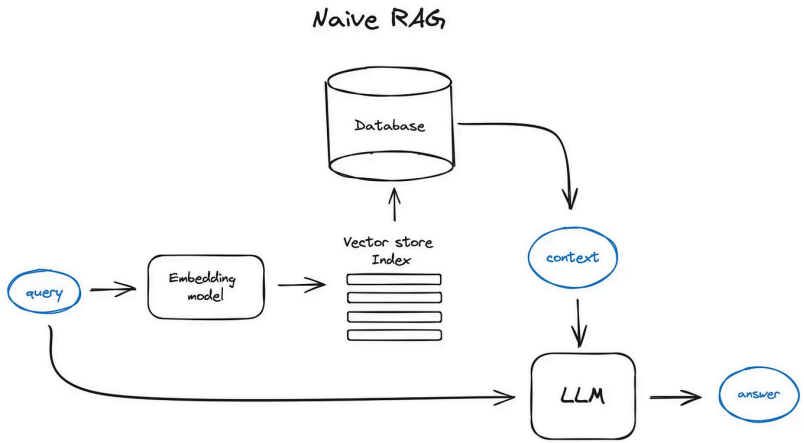
- Retriever fetches relevant passages from a knowledge base (vector DB, search engine).
- Passages are concatenated with the query, fed into the LLM.

Advantages:

- Simple, effective when the answer is well-covered in the KB.
- Easy to implement with existing tools.

Limitations:

- Relies on near-exact matches in retrieval.
- Struggles with under-specified or nuanced queries.



Credit: Ivan Ilin

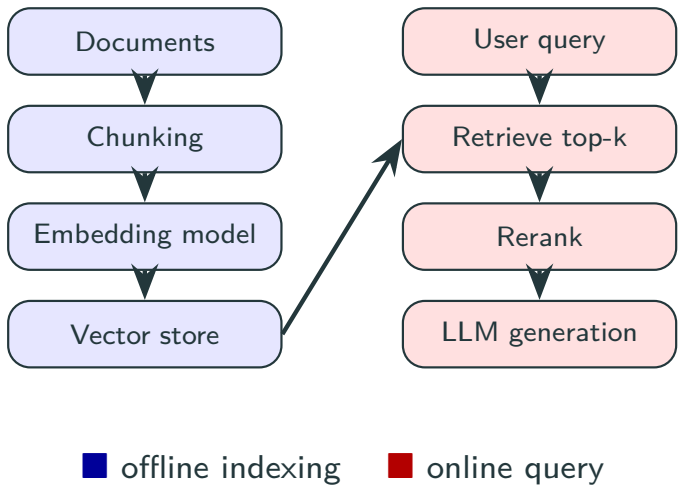
The Full RAG Pipeline

Offline indexing:

- **Chunk** documents (size + overlap).
- **Embed** each chunk with an embedding model.
- Store vectors in a **vector store** (FAISS, pgvector).

Online query:

- **Embed** the query, **retrieve** top-k nearest chunks.
- **Rerank** to keep the most relevant few.
- **Generate**: inject context into the prompt.



Key knobs: **chunk size & overlap**, **embedding model**, **top-k**, **reranker**, **prompt**.

Reranking: Cheap Recall, then Precise Ranking

Why a second stage? The first retriever (bi-encoder / ANN search) is fast but coarse: it embeds query and document **separately**.

- **Stage 1, Retrieve (recall):** pull top-50/100 candidates cheaply from the vector store.
- **Stage 2, Rerank (precision):** a **cross-encoder** reads (query, document) *together* and scores true relevance, keeping the best top-3/5.
- Cross-encoders are more accurate but too slow for the whole corpus $\Rightarrow$ retrieve-then-rerank is the standard compromise.
- Fewer, better chunks $\Rightarrow$ less noise, lower cost, fewer hallucinations.

e.g. cross-encoder rerankers (MS-MARCO MiniLM), Cohere Rerank, ColBERT late-interaction.

Evaluating a RAG System

Two failure points ⇒ two things to measure.

1. **Retrieval quality** (right chunks?), vs. labelled relevant passages:
   - **Recall@k / Precision@k:** fraction of relevant chunks in the top-k / fraction of the top-k that are relevant.
   - **MRR** (Mean Reciprocal Rank): average of 1/rank of the *first* relevant chunk; rewards ranking a good chunk high.
   - **nDCG** (normalized Discounted Cumulative Gain): relevance scores **discounted by rank** ($1/\log_2(i{+}1)$), normalized by the ideal ordering.
2. **Generation quality** (answer used them correctly?):
   - **Faithfulness / groundedness:** every claim supported by the retrieved context? (anti-hallucination)
   - **Answer relevance** + **context precision/recall**: context sufficient and not noisy?

Frameworks: **RAGAS**, TruLens, ARES; many use an **LLM-as-a-judge** (Session 10).

Advanced Retrieval: HyDE & Query Transformation

When naive retrieval fails on vague or abstract queries, transform the *query side*:

- **Query rewriting / expansion:** reformulate or split the question before retrieving.
- **HyDE** (Gao et al., 2022): let the LLM **imagine** a hypothetical answer, embed *that*, and retrieve documents close to it in meaning.
- Bridges the gap between a short query and the longer documents that answer it.

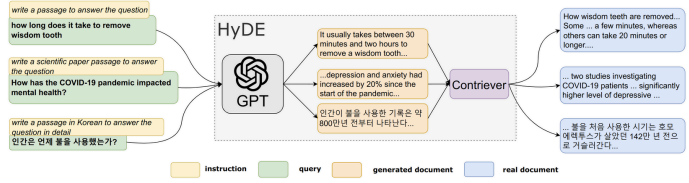


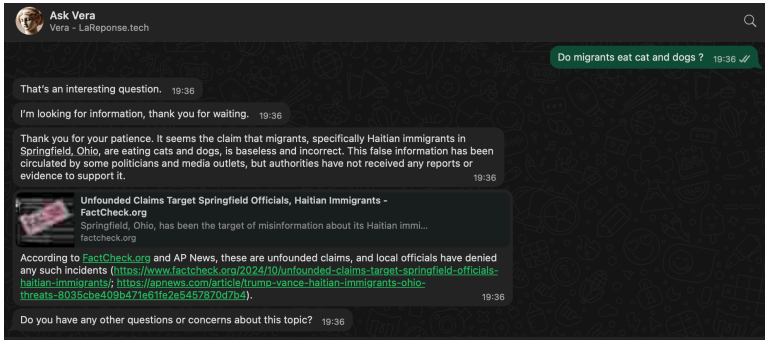
Figure 1: An illustration of the HyDE model. Documents snippets are shown. HyDE serves all types of queries without changing the underlying GPT-3 and Contriever/mContriever models.

HyDE, Gao et al. (2022)

Useful, but optional: a good reranker often closes most of the gap with less complexity.

How Vera Works: RAG for Trusted Fact-Checking

- **Query:** user submits a question or claim.
- **Retriever:** web search over a **curated corpus**: factcheckers (AFP Factuel, les décodeurs) and verified news media (Le Monde).
- **Reranker:** re-scores sources by trustworthiness and relevance, keeping only the best few.
- **Generator:** LLM grounds its response in the reranked sources.
- **Attribution:** Vera cites its sources for transparency and traceability.



Credit: Whatsapp

Why it works: focusing on **trusted, factual sources** reduces hallucination and misinformation risk.

Fine-Tuning

Two Kinds of Fine-Tuning: Knowledge vs. Behavior

- **Continual / Continued Pre-Training (CPT):** keep doing **next-token prediction** on raw domain text to **inject knowledge** (a new language, legal/medical/code corpus).
- **Supervised Fine-Tuning (SFT):** train on **prompt** $\rightarrow$ **response** pairs to shape **behavior** (format, tone, instruction following).

Rule of thumb:

- "The model doesn't *know* my domain" $\Rightarrow$ CPT (lots of unlabeled text).
- "The model knows it but doesn't *answer* the way I want" $\Rightarrow$ SFT (fewer labeled pairs).
- Typical recipe: **CPT** $\rightarrow$ **SFT** $\rightarrow$ **preference optimization (DPO)**.

Continued Pre-Training: What You Actually Do

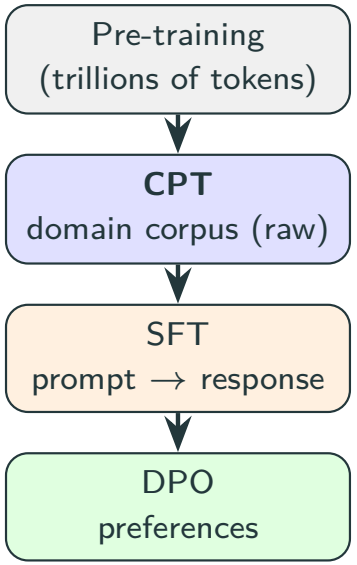
Concretely: take a base model and **keep pre-training** it on a large *raw* domain corpus, with the **same next-token prediction** loss (no labels, no prompt/response format).

What it teaches: new **vocabulary** (legal, medical, code, a new language) and domain **context**, what usually follows what.

What it is for: fill a **knowledge gap** the base model never saw, *before* SFT teaches it how to answer.

It works in practice:

- **Code Llama**: Llama 2 + 500B tokens of code.
- **Llemma**: Code Llama + 200B math tokens (Proof-Pile-2).



What is 1B tokens?

≈ 750M words ≈ **7,500 books** (or all of English Wikipedia ≈ 4–5B).

Continued Pre-Training: How To Do It Well

Main risk: catastrophic forgetting, gaining the new domain while losing general ability.

Protocol (Ibrahim/Gupta et al., 2024):

- **Re-warm then re-decay** the learning rate (fresh cosine), don't resume the old one.
- **Replay old data:** mix in $\approx$5% of the original distribution to preserve prior skills.
- LR-reset + replay matches full re-training at a fraction of the compute.

Data mixture matters (Chen et al., ACL 2025, Llama-3-SynE):

- Bilingual adaptation used a **1:7:2** mix (new-lang : English : synthetic).
- Perplexity-based **curriculum** (easy $\rightarrow$ hard); add synthetic code/QA, which degrades fast.

Continued Pre-Training: How Many Tokens?

Two reference points to size the corpus:

- **Vs. the base dataset:** a **small fraction** of the original pre-training tokens, typically a **few percent** (∼1–10%). Chen et al. (2025) run rounds of ≈**40B tokens**, vs. **trillions** for pre-training.
- **Vs. model weights:** pre-training sits near **Chinchilla ≈20 tokens/param** (Hoffmann et al., 2022). Domain CPT runs **well below**: for a 7–8B model, tens of B tokens ≈ **a few tokens/param**.

In practice, by model size:

- **7B**: ∼30–200B tokens (SaulLM 30B, Meditron 47B, Llemma 200B).
- ∼**24–34B**: ∼200–500B tokens (Code Llama 34B: 500B).
- **70B**: ∼0.5–1T tokens (Code Llama 70B: 1T).

Too few ⇒ no knowledge gain; **too many** ⇒ forgetting + wasted compute.
Track **val perplexity per topic** to decide when to stop.

Supervised Fine-Tuning (SFT)

Overview of Supervised Fine-Tuning: leverages labeled data to enhance models' understanding & response quality in targeted domains.

Examples of Open Source Datasets:

- **OASST1/2:** OpenAssistant multilingual conversation trees, human-written and annotated for dialog quality.
- **UltraChat / UltraFeedback:** large-scale synthetic instruction and preference data, widely used to align open models (e.g. Zephyr).
- **Tulu / FLAN collections:** curated instruction mixtures spanning many task types for broad instruction following.

Significance: Supervised FT enables LLMs like ChatGPT to perform better in specialized tasks or languages, making them more versatile and effective in real-world applications.

Fine Tuning dataset example

prompt (string)	response (string)
" Good morning I have a Wpf datagrid that is displaying an observable collection of a custom…	"One possible solution is to use a fixed width for the GroupItem header and align the header and the…
"<h2>Hi, How can I generate a pdf with the screen visual data, or generate a pdf of the data being…	"To generate a PDF with the screen visual data, you can use a library such as pdf. Here's an example:…
"<pre><code>package com.kovair.omnibus.adapter.platform; import…	"The issue might be related to class loading and garbage collection. When a class loader loads a…
" I'm trying to get it so that all of the items in ListView.builder can be displayed on the screen…	"To make the whole page scrollable, remove the `SingleChildScrollView` and wrap the entire…
" I have used a <code>ListView</code> and the parent in the <code>xml</code> is…	"The issue seems to be with the layout parameters being set in the `getView()` method. The code is…
" I am calling a stored proc [MS SQL] using EF5 from a .net application The call from EF</p…	" This is likely due to the fact that CHAR columns are fixed-length and padded with spaces to…
" This code is about viewing a published consultation schedule. Unfortunately I'm stuck wit…	"The issue with the if statement is that it is inside the for loop, but it should be outside of…

GPT-4all Dataset

Implementing Supervised Fine-Tuning with SFTTrainer

Using Hugging Face's 'SFTTrainer':

The 'SFTTrainer' is a tool in Hugging Face's Transformers library designed to streamline the fine-tuning of large language models on specific datasets.

Python Code Example

```
from transformers import AutoTokenizer, AutoModelForCausalLM
from trl import SFTTrainer, SFTConfig
from datasets import load_dataset

model_id = "meta-llama/Llama-3.2-3B"
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(model_id)

train_dataset = load_dataset("path/to/dataset", split="train")
```

Implementing Supervised Fine-Tuning with SFTTrainer

Python Code Example

```
# Define training config
training_args = SFTConfig(
    output_dir="./fine_tuned_model",
    num_train_epochs=3,
    per_device_train_batch_size=4,
    logging_dir="./logs",
)

# Initialize SFTTrainer
trainer = SFTTrainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    processing_class=tokenizer,
)

trainer.train()
```

Limitations of Supervised Fine-Tuning without RHLF

Challenges: While SFT can significantly enhance LLMs' performance on specific tasks, doing so without access to RLHF introduces several limitations:

- **Bias Amplification:** Fine-tuning on biased datasets without RHLF can lead to the amplification of existing biases, affecting the fairness and neutrality of the model's outputs.
- **Overfitting:** The lack of diverse human feedback during fine-tuning may result in models that overfit to the training data, hindering their generalization to unseen contexts.
- **Missed Learning Opportunities:** RLHF can provide unique insights and corrections that are crucial for improving models. Without it, models miss out on learning from complex human interactions and corrections.

Conclusion: Incorporating RLHF is crucial for developing more adaptable, unbiased, and generalizable LLMs. Overcoming these limitations requires innovative approaches to integrate human feedback effectively.

Training Complexity of a 7B Parameter Model

Resource Requirements:

- **Model size:** $\approx$ 28 GB in 32-bit (fp32) precision.
- **Training memory:** Gradient storage + optimizer states increase the memory footprint $\approx$ 3-4x.
- **Total memory footprint:** $\approx$ 140 GB.

GPU Requirements:

- Requires multiple high-end GPUs (A100 80GB, H100 80GB, etc.).
- **Example setup:** 2$\times$ A100 80 GB or 4$\times$ A100 40 GB GPUs for data/gradient parallelism.

Time Complexity:

- Days to weeks of training depending on dataset size and compute budget.
- Infeasible on a single GPU.

LoRA: Train Large Models on a Single GPU

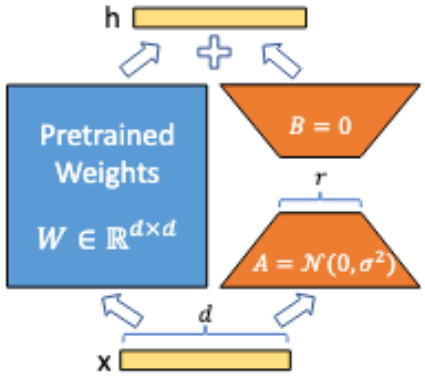
- LoRA stands for **Low-Rank Adaptation**.
- It lets you fine-tune LLM using small, trainable adapters instead of updating the whole model.

How does it work?

- Original model weights are **frozen**.
- Add two small matrices, A and B, to capture the task-specific adaptation.
- Only these small matrices are trained.

Why is it great?

- Needs much less memory and compute.
- You can fine-tune a huge model (e.g., 7B LLaMA) on a single 24 GB GPU.



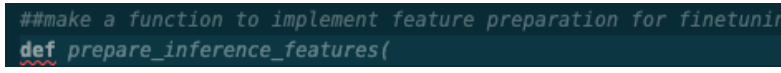
LoRA matrices.

Credit: Hu et al. (2021)

RLHF through UX: Copilot Completion - Part 1

Integrating Human Feedback with UX in LLMs: Advanced UX mechanisms, like those in GitHub Copilot, demonstrate the power of integrating human feedback into LLM fine-tuning:

- **Effortless Feedback Integration:**
  - Direct workflow integrations for accepting suggestions (e.g., using TAB).
  - Navigation shortcuts for efficient suggestion review.



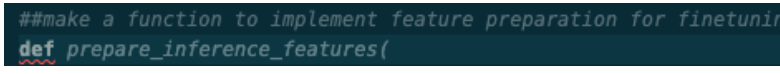
GitHub Copilot UX

RLHF through UX: Copilot Completion - Part 2

User-Friendly Interaction & Optimized Feedback:

- **User-Friendly Interaction:**
  - Suggestions are passive, maintaining user workflow integrity.
  - High latency sensitivity for timely, relevant suggestions.
- **Optimized Feedback for Fine-Tuning:**
  - Capturing implicit signals from user interactions for model improvement.
  - Encourages user engagement through low requirements and high incentives.

Impact: UX designs that facilitate easy human feedback collection empower continuous LLM learning and user satisfaction.



GitHub Copilot UX: Feedback

QA and Takeaways

QA

Open Discussion

- Feel free to ask questions or share your thoughts about today's topics.
- Any insights, experiences, or perspectives you'd like to discuss are welcome.

Summary of Key Takeaways

- **LLMs & training:** evolution since 2022; aligned via RLHF/PPO, then DPO and its variants.
- **Reasoning models:** trained with RL on *verifiable* rewards (GRPO); learned chain of thought.
- **Prompt engineering:** few-shot, chain-of-thought, structured outputs; tune temperature *or* top-p.
- **RAG:** full pipeline (chunk $\rightarrow$ embed $\rightarrow$ retrieve $\rightarrow$ rerank $\rightarrow$ generate), with reranking and evaluation.
- **Fine-tuning:** continued pre-training (knowledge) vs. SFT (behavior); LoRA and implicit RLHF via UX.