

CME 295: Transformers & Large Language Models

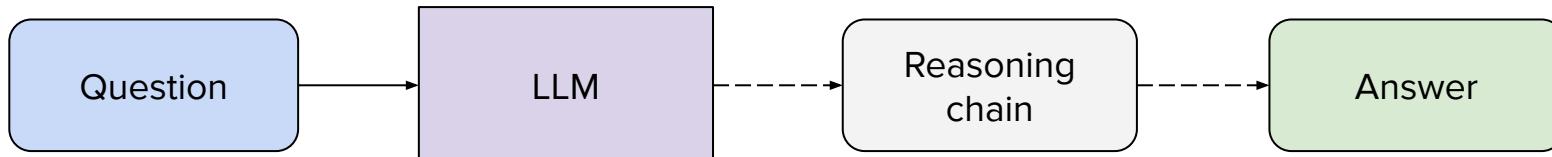


Afshine Amidi & Shervine Amidi



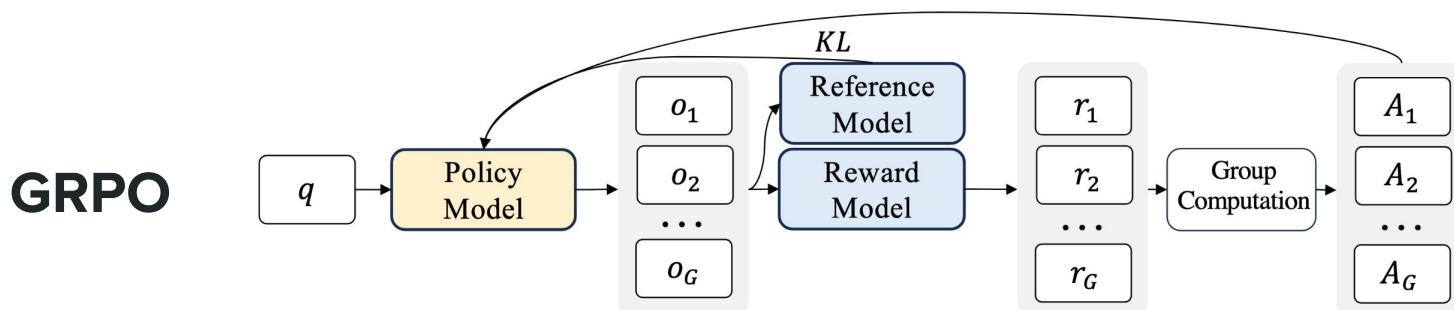
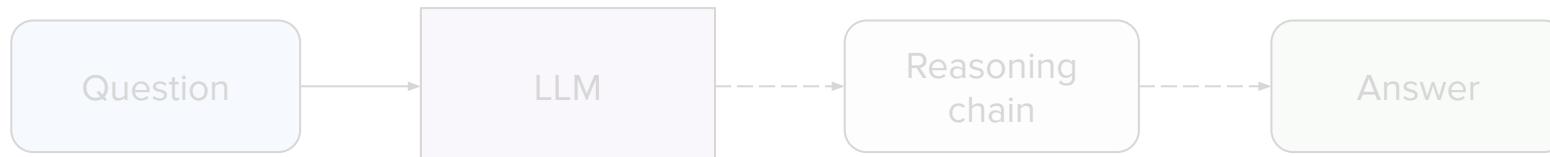
Recap of last episode...

Reasoning models

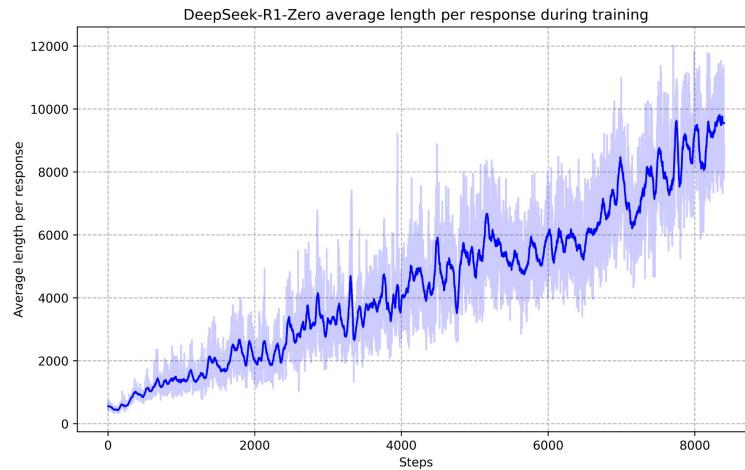
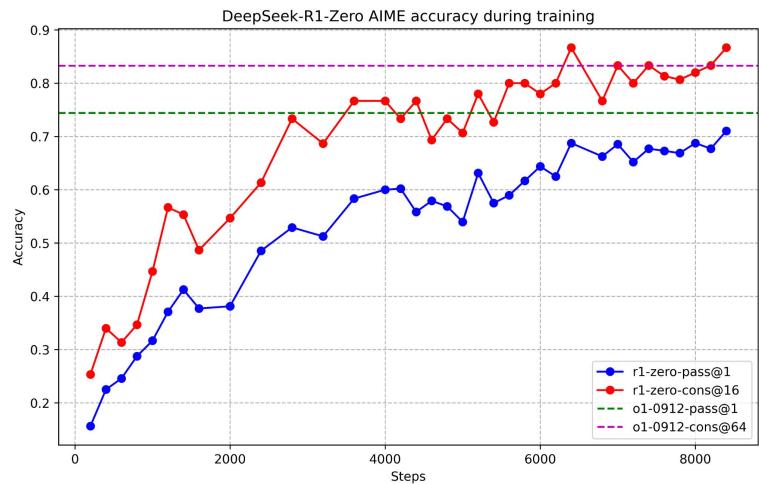


Recap of last episode...

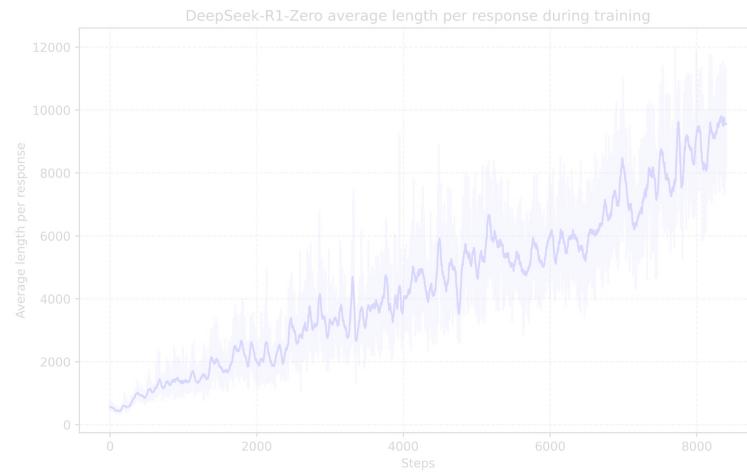
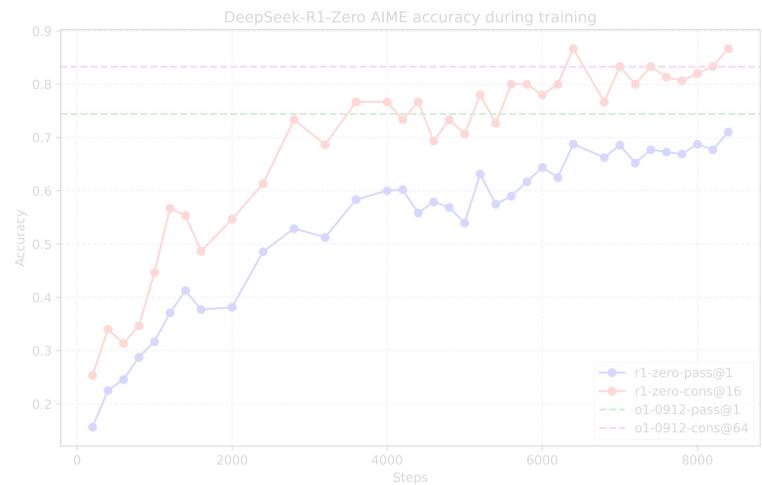
Reasoning models



Recap of last episode...

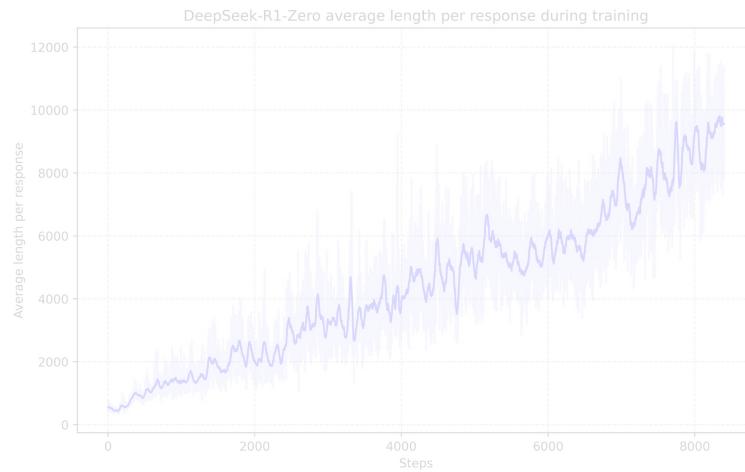
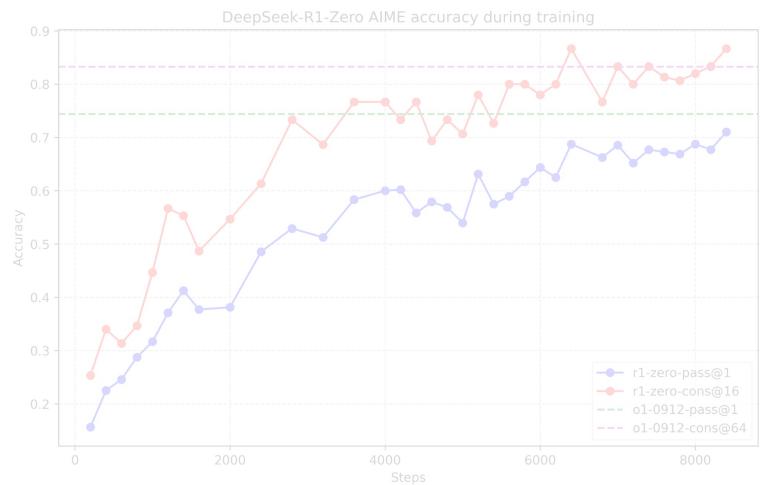


Recap of last episode...



Original $\frac{1}{G} \sum_{i=1}^G \frac{1}{|\mathbf{o}_i|} \sum_{t=1}^{|\mathbf{o}_i|}$

Recap of last episode...



Original

$$\frac{1}{G} \sum_{i=1}^G \frac{1}{|\mathbf{o}_i|} \sum_{t=1}^{|\mathbf{o}_i|}$$

DAPO

$$\frac{1}{\sum_{i=1}^G |\mathbf{o}_i|} \sum_{i=1}^G \sum_{t=1}^{|\mathbf{o}_i|}$$

Dr. GRPO

$$\frac{1}{G} \sum_{i=1}^G \sum_{t=1}^{|\mathbf{o}_i|}$$

"Understanding R1-Zero-Like Training: A Critical Perspective", Liu et al., 2025.

"DAPO: An Open-Source LLM Reinforcement Learning System at Scale", Yu et al., 2025.

Recap of last episode...

Strengths.

- Great at imitation or idea generation
- Amazing at generating or debugging code

Weaknesses.

- Limited reasoning
- Knowledge is static
- Cannot perform actions
- Hard to evaluate

Recap of last episode...

Strengths.

- Great at imitation or idea generation
- Amazing at generating or debugging code

Weaknesses.

- Limited reasoning
- Knowledge is static
- Cannot perform actions
- Hard to evaluate



Focus of last lecture

Recap of last episode...

Strengths.

- Great at imitation or idea generation
- Amazing at generating or debugging code

Weaknesses.

- Limited reasoning
 - **Knowledge is static** ←
 - **Cannot perform actions** ←
 - Hard to evaluate
- Focus of today**
-
- ```
graph LR; A[Knowledge is static] --> B[Focus of today]; C[Cannot perform actions] --> B;
```



# Transformers & Large Language Models

RAG—  
*it is being used for updating the latest  
information in LLM.  
[Retrieval Augmented Generation]*

Tool calling

Agents

# Motivation

- Knowledge of LLM constrained to pretraining data

↓  
i.e. the model which is being trained upto the  
cut-off date, after this cut-off date model  
will not be able to respond anything beyond  
this date. So, RAG (Retrieval Augmented Generative)  
can help to update the information after the cut-off.

# Motivation

- Knowledge of LLM constrained to pretraining data

The screenshot shows the GPT-5 model card from OpenAI. At the top, there is a large '5' icon, the text 'GPT-5 Default', a dropdown menu, and a 'Try in Playground' button. Below this, a subtext states 'The best model for coding and agentic tasks across domains'. The card is divided into five main sections: 'REASONING' (Higher, 4 icons), 'SPEED' (Medium, 3 lightning bolt icons), 'PRICE' (\$1.25 + \$10, Input + Output), 'INPUT' (Text, image, 4 icons), and 'OUTPUT' (Text, 4 icons). A large 'Compare' button is located to the right of the input section.

| REASONING | SPEED  | PRICE                           | INPUT       | OUTPUT |
|-----------|--------|---------------------------------|-------------|--------|
| Higher    | Medium | \$1.25 + \$10<br>Input + Output | Text, image | Text   |

GPT-5 is our flagship model for coding, reasoning, and agentic tasks across domains. Learn more in our [GPT-5 usage guide](#).

- ❖ 400,000 context window
- 128,000 max output tokens
- ⌚ Sep 30, 2024 knowledge cutoff *knowledge cutoff date*  
↳ *i.e. if we are querying after this date, LLM will not be able to respond.*
- ⌚ Reasoning token support

# Motivation

- Knowledge of LLM constrained to pretraining data

The screenshot shows the GPT-5 model card from OpenAI. At the top, there is a large '5' icon, the text 'GPT-5 Default', a dropdown menu, and a 'Try in Playground' button. Below this, a subtext states 'The best model for coding and agentic tasks across domains'. The card is divided into five main sections: 'REASONING' (Higher, 4 icons), 'SPEED' (Medium, 3 lightning bolt icons), 'PRICE' (\$1.25 + \$10, Input + Output), 'INPUT' (Text, image, 4 icons), and 'OUTPUT' (Text, 4 icons). A detailed description below the card reads: 'GPT-5 is our flagship model for coding, reasoning, and agentic tasks across domains. Learn more in our [GPT-5 usage guide](#).'

GPT-5 is our flagship model for coding, reasoning, and agentic tasks across domains. Learn more in our [GPT-5 usage guide](#).

❖ 400,000 context window

→ 128,000 max output tokens

⌚ Sep 30, 2024 knowledge cutoff

⌚ Reasoning token support

as we have the context length limited, we can't just add additional token for training after the cut-off date.

# Motivation

- Knowledge of LLM constrained to pretraining data
- Limited context size

# Motivation

- Knowledge of LLM constrained to pretraining data
- Limited context size

The screenshot shows the GPT-5 model card from OpenAI. At the top, there is a colorful icon with the number '5', the text 'GPT-5' in bold, a 'Default' dropdown, and a copy icon. Below this, a subtext states 'The best model for coding and agentic tasks across domains'. To the right are two buttons: 'Compare' and 'Try in Playground'. The main section contains five performance metrics: Reasoning (Higher, represented by four circles), Speed (Medium, represented by three lightning bolts), Price (\$1.25 + \$10, Input + Output), Input (Text, image, represented by icons), and Output (Text, represented by icons). Below these metrics is a descriptive paragraph: 'GPT-5 is our flagship model for coding, reasoning, and agentic tasks across domains. Learn more in our [GPT-5 usage guide](#)'. To the right of this paragraph is a red-bordered box containing four key features: '400,000 context window', '128,000 max output tokens', 'Sep 30, 2024 knowledge cutoff', and 'Reasoning token support'.

GPT-5 Default ⚪  
The best model for coding and agentic tasks across domains

Compare Try in Playground

| REASONING | SPEED  | PRICE                           | INPUT       | OUTPUT |
|-----------|--------|---------------------------------|-------------|--------|
| Higher    | Medium | \$1.25 + \$10<br>Input + Output | Text, image | Text   |

GPT-5 is our flagship model for coding, reasoning, and agentic tasks across domains. Learn more in our [GPT-5 usage guide](#).

◆ 400,000 context window  
→ 128,000 max output tokens  
📅 Sep 30, 2024 knowledge cutoff  
⌚ Reasoning token support

# Motivation

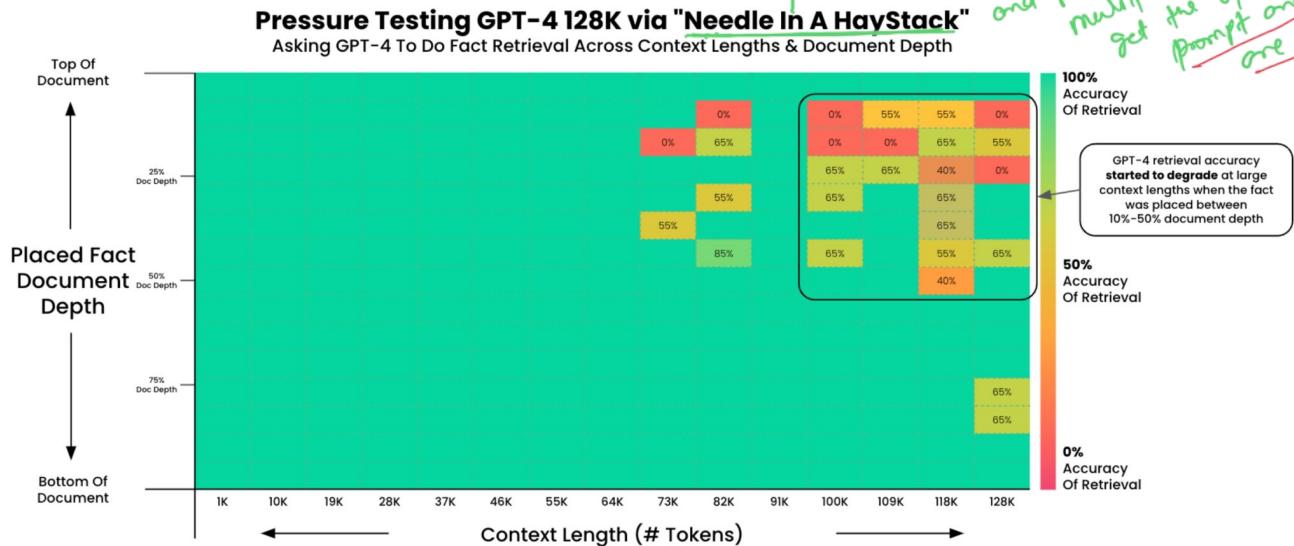
- Knowledge of LLM constrained to pretraining data
- Limited context size
- LLM gets **distracted** by **useless** information



adding irrelevant information  
to the model also creates  
problems and may cause the  
model to generate the output improperly

# Motivation

- Knowledge of LLM constrained to pretraining data
- Limited context size
- LLM gets **distracted** by **useless** information

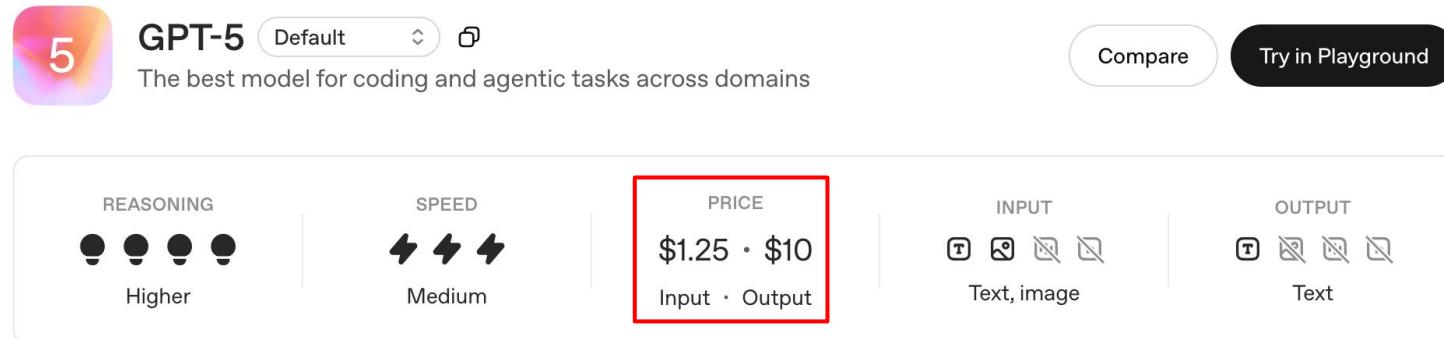


# Motivation

- Knowledge of LLM constrained to pretraining data
- Limited context size
- LLM gets **distracted** by **useless** information
- **Pricing is per** input/output token

# Motivation

- Knowledge of LLM constrained to pretraining data
- Limited context size
- LLM gets **distracted** by **useless** information
- **Pricing is per input/output token**



# Overview

## RAG = Retrieval-Augmented Generation

↓  
so, instead of putting all the information for the LLM to  
be updated after cutoff date, RAG supports to extract  
only the relevant information to add/inject to the model to get  
updated after cut-off date of LLM.

# Overview

**RAG** = Retrieval-Augmented Generation

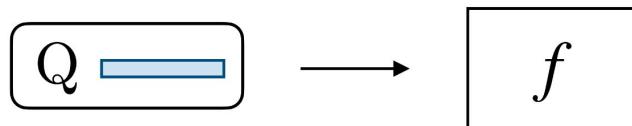
**Idea.** Augment prompt with **relevant** pieces of information.

↓  
Q: How do we get only relevant information?

# Overview

**RAG** = Retrieval-Augmented Generation

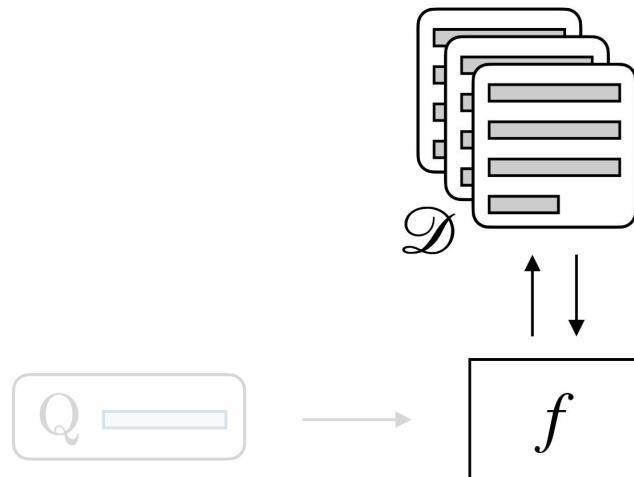
**Idea.** Augment prompt with **relevant** pieces of information.



# Overview

**RAG** = Retrieval-Augmented Generation

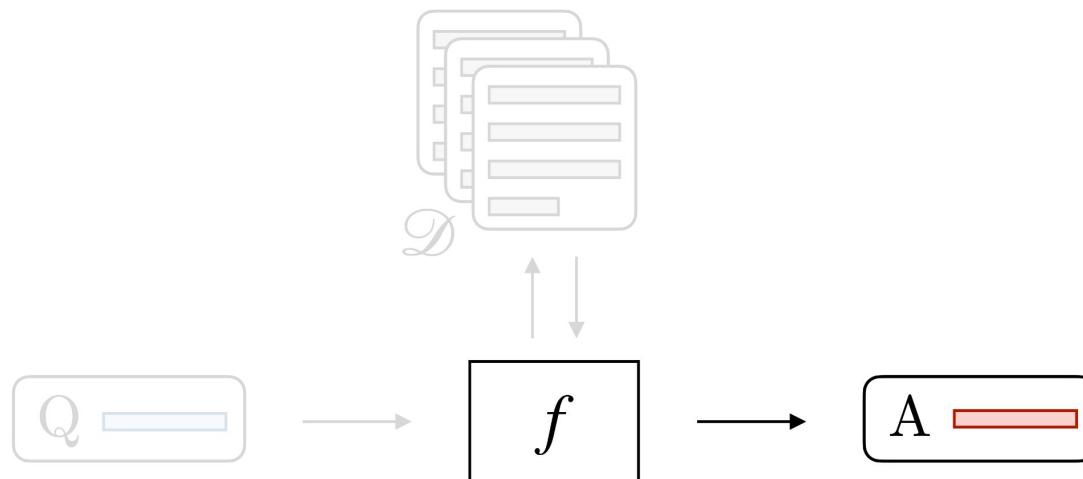
Idea. Augment prompt with **relevant** pieces of information.



# Overview

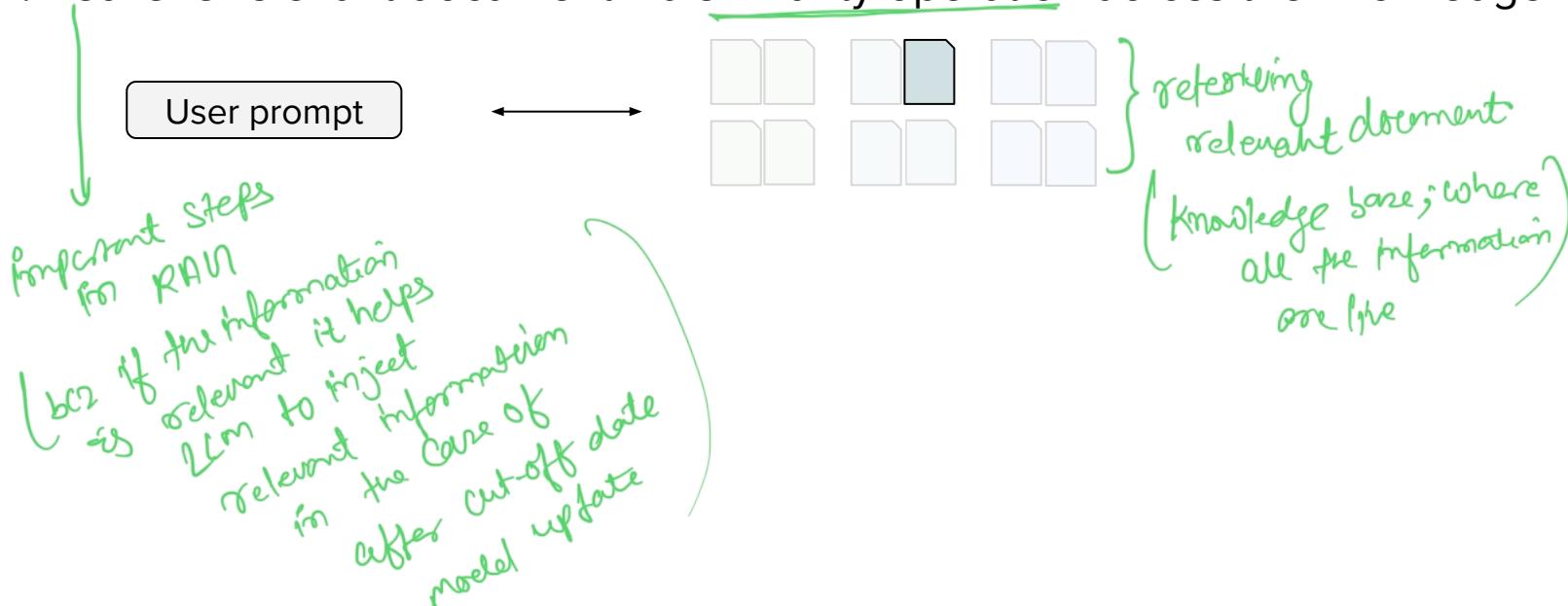
**RAG** = Retrieval-Augmented Generation

Idea. Augment prompt with **relevant** pieces of information.



# Retrieve, Generate, Augment

## 1. Retrieve relevant document via similarity operation across the knowledge base



# Retrieve, Generate, Augment

1. **Retrieve** relevant document via similarity operation across the knowledge base



2. **Augment** prompt with retrieved information



# Retrieve, Generate, Augment

1. **Retrieve** relevant document via similarity operation across the knowledge base



2. **Augment** prompt with retrieved information



3. **Generate** response



feed the prompt  
(relevant information prompt)

# Retrieve, Generate, Augment

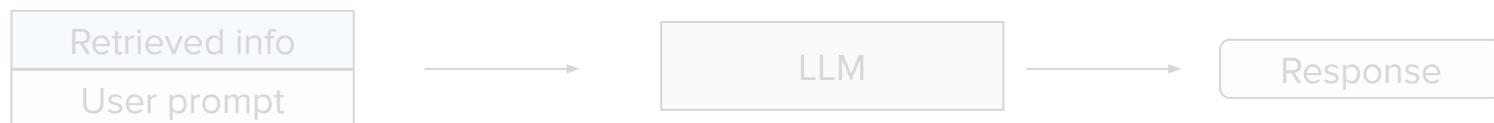
1. **Retrieve** relevant document via similarity operation across the knowledge base



2. **Augment** prompt with retrieved information

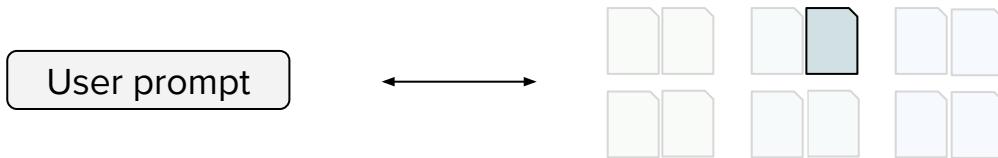


3. **Generate** response



# Focus on retrieval stage

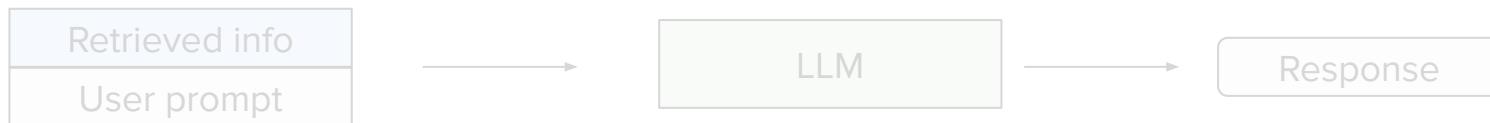
1. **R**etrieve relevant document via similarity operation across the knowledge base



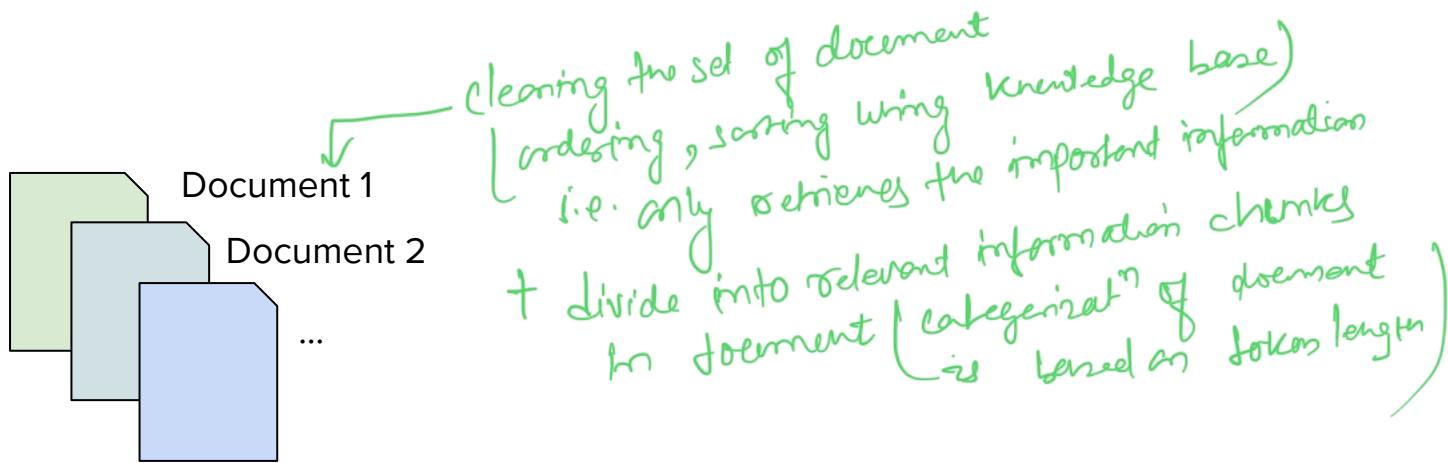
2. **A**ugment prompt with retrieved information



3. **G**enerate response

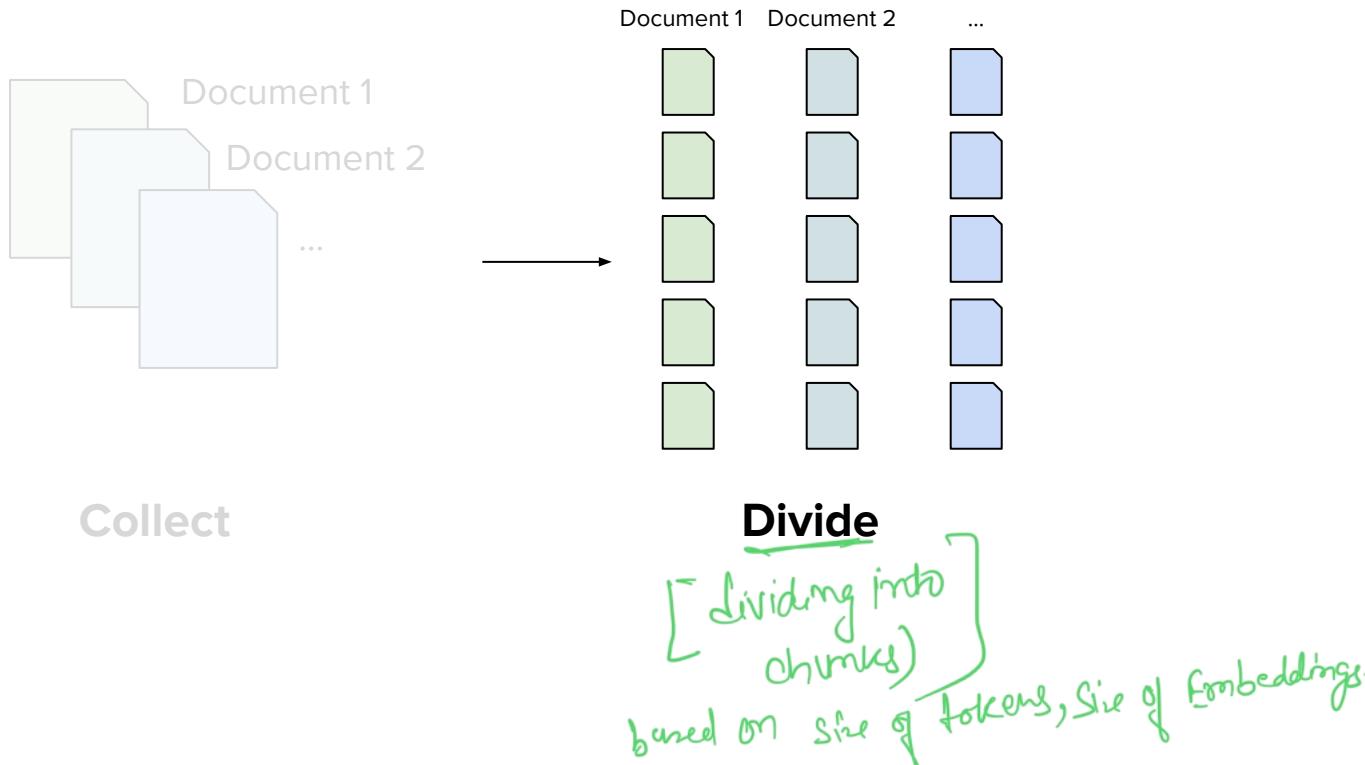


# Prerequisite: Create knowledge base

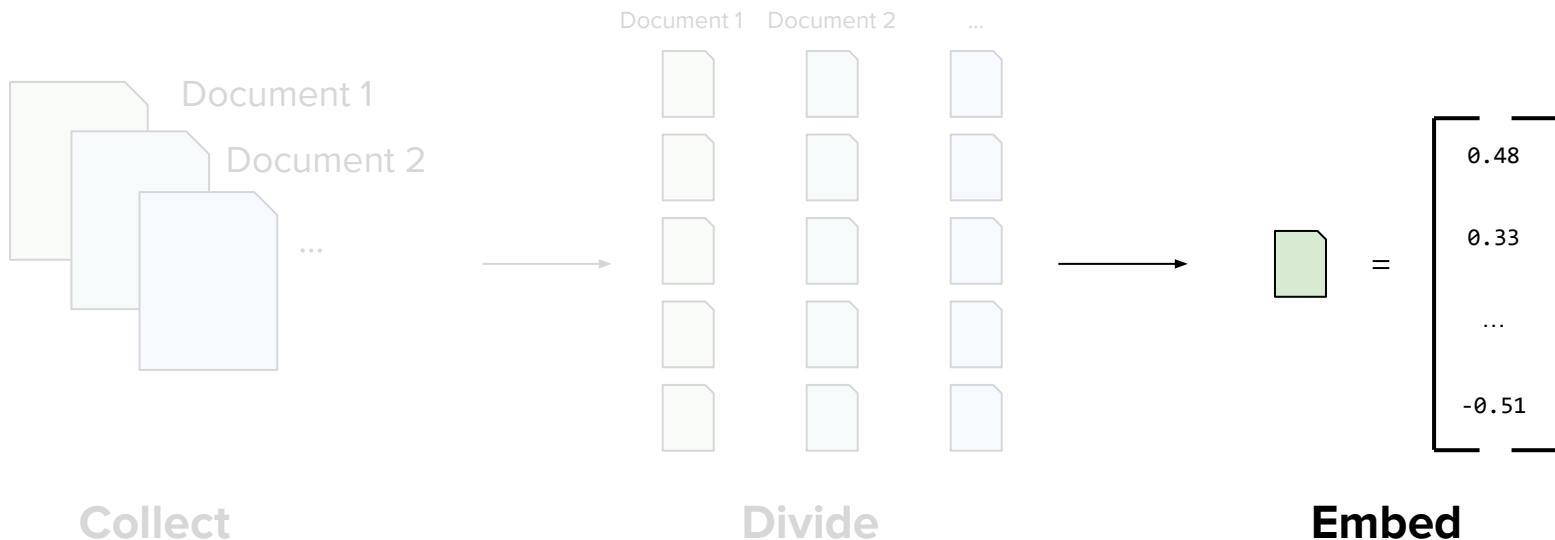


## Collect

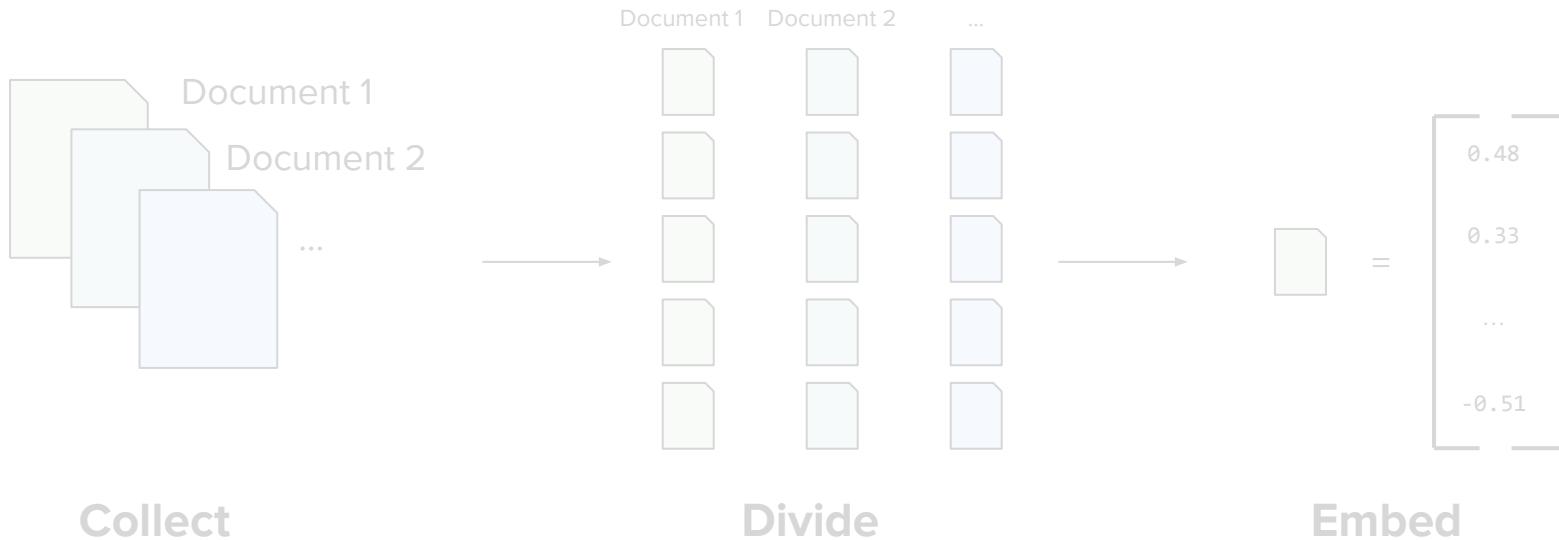
# Prerequisite: Create knowledge base



# Prerequisite: Create knowledge base



# Prerequisite: Create knowledge base



**Hyperparameters.** Embedding size, chunk size, overlap between chunks

↓  
how big is your  
piece of documents are?

↓  
how much overlap  
b/w  
chunks?

# Retrieval overview



→ go through many-many number of chunks and filter down much smaller set of potentially relevant candidates.

## Step 1 – Candidate retrieval: Select potentially-relevant candidates

- ✓ Maximize recall → to select relevant candidates, need to have maximum recall
- ✓ Semantic embeddings and optionally keyword-based methods

# Retrieval overview



## Step 1 – Candidate retrieval: Select potentially-relevant candidates

- Maximize recall
- Semantic embeddings and optionally keyword-based methods

ranking all the documents w.r.t. relevance on relevant documents.

100

## Step 2 – Ranking: Give final relevance score

- Maximize precision
- Re-ranking on smaller set of candidates

Step 1: candidate retrieval - fitting down the potentially huge knowledge base  
to somehow filter down to relevant #  
of candidates.

**Objective.** Select potentially-relevant candidates via search across knowledge base

# Step 1: candidate retrieval

**Objective.** Select potentially-relevant candidates via search across knowledge base

 **Method 1.** Semantic search using **embeddings-based similarity**

# Step 1: candidate retrieval

**Objective.** Select potentially-relevant candidates via search across knowledge base

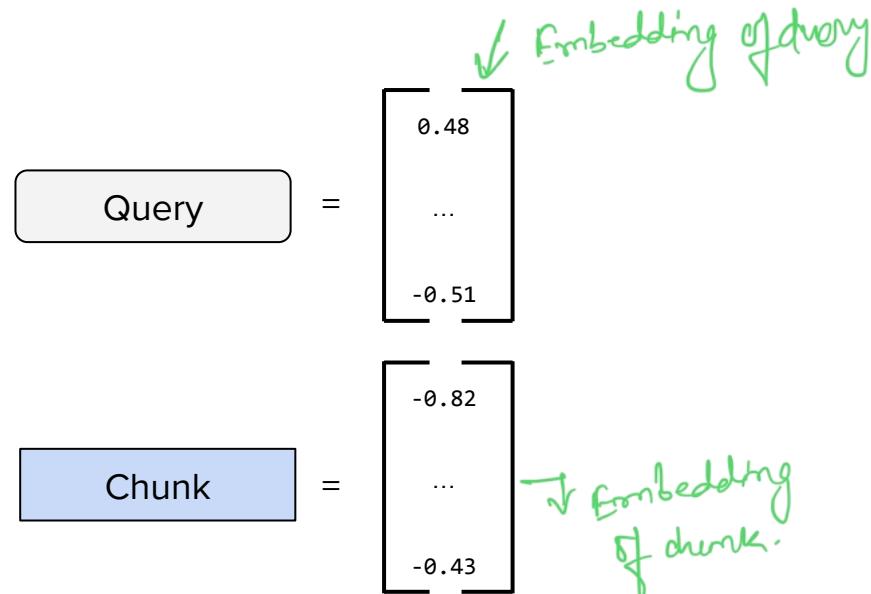
**Method 1.** Semantic search using **embeddings-based similarity**



# Step 1: candidate retrieval

**Objective.** Select potentially-relevant candidates via search across knowledge base

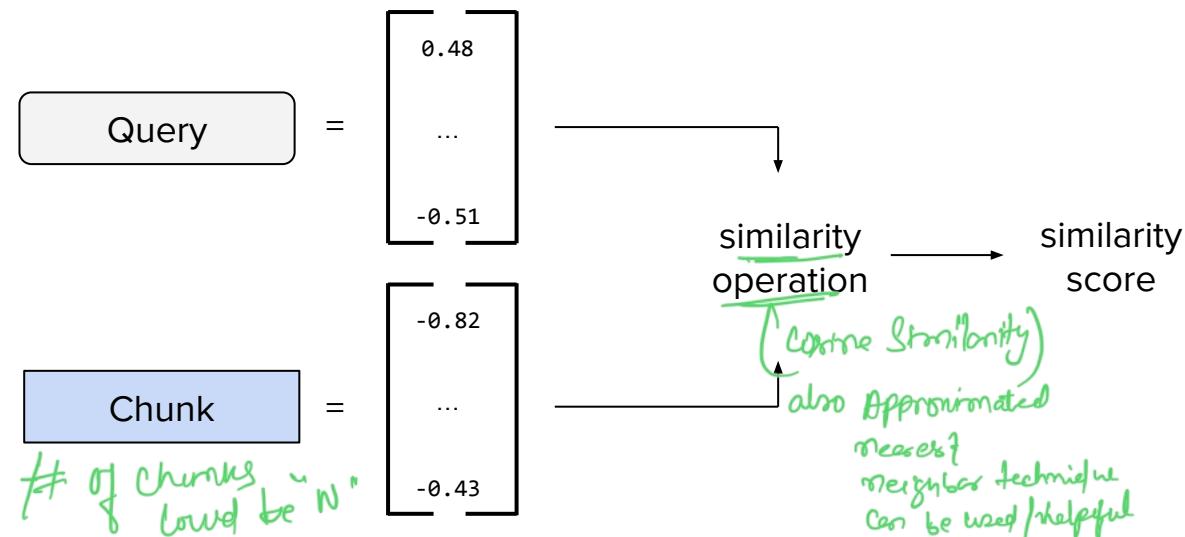
**Method 1.** Semantic search using **embeddings-based similarity**



# Step 1: candidate retrieval

**Objective.** Select potentially-relevant candidates via search across knowledge base

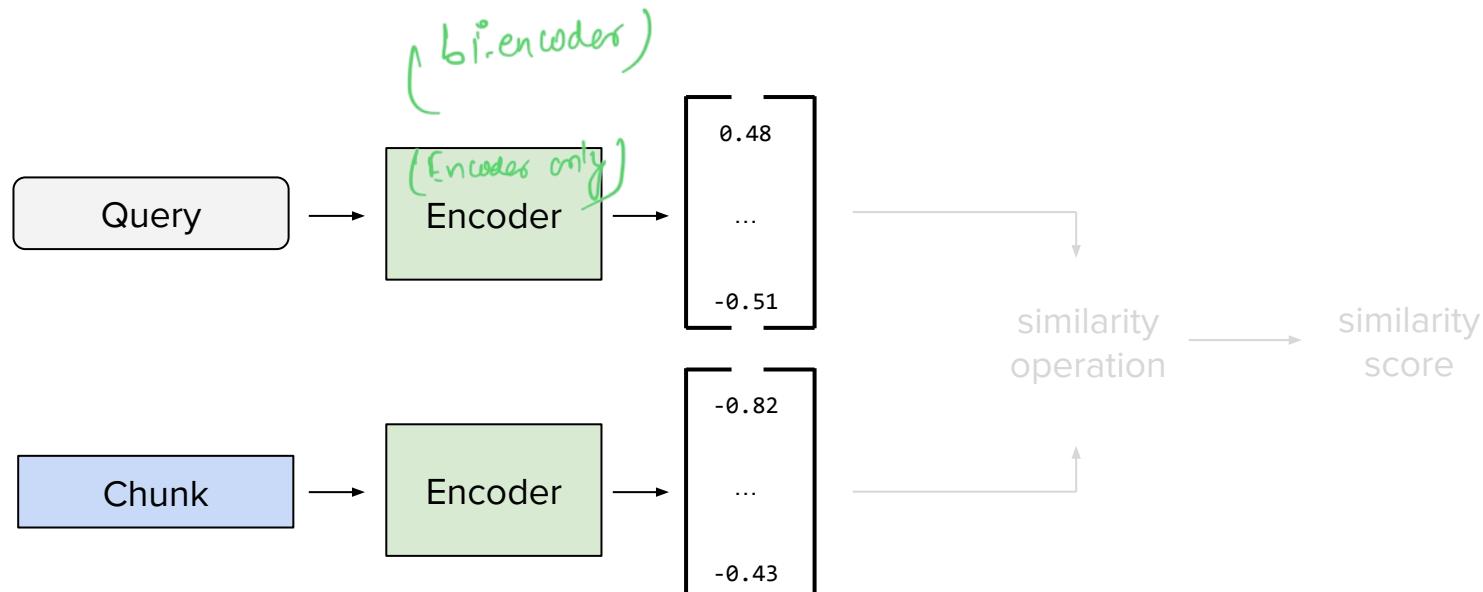
**Method 1.** Semantic search using **embeddings-based similarity**



# Step 1: candidate retrieval

**Objective.** Select potentially-relevant candidates via search across knowledge base

**Method 1.** Semantic search using **embeddings-based similarity**

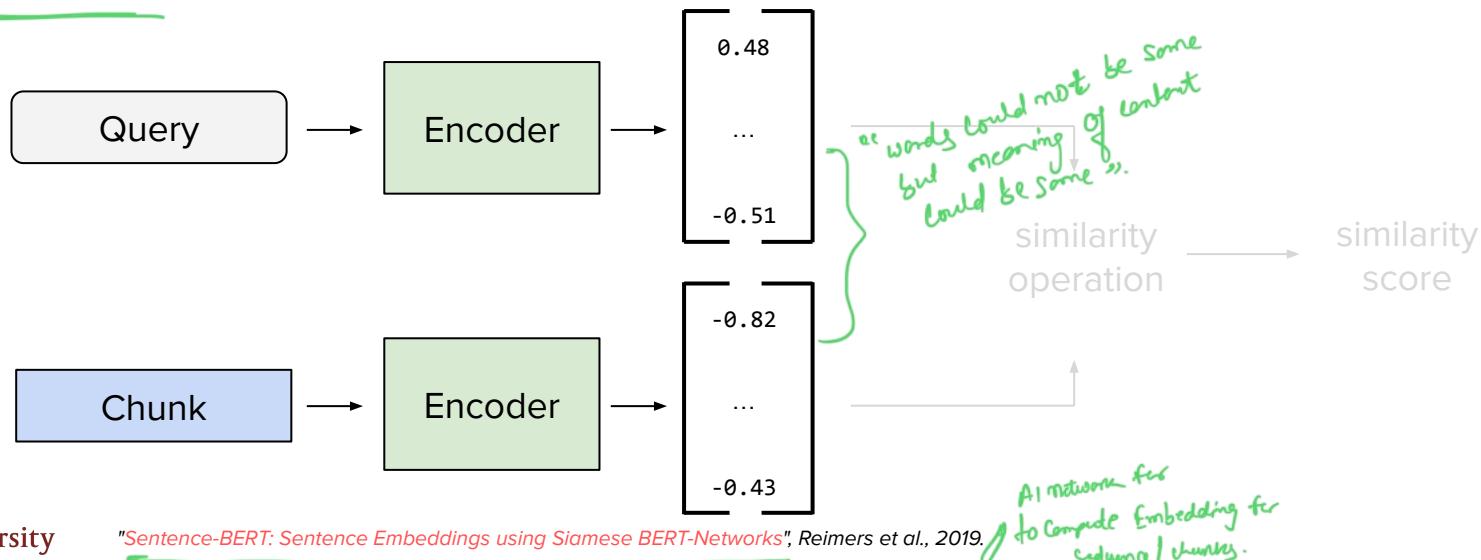


# Step 1: candidate retrieval

**Objective.** Select potentially-relevant candidates via search across knowledge base

**Method 1.** Semantic search using **embeddings-based similarity**

"bi-encoder"



# Step 1: candidate retrieval

**Objective.** Select potentially-relevant candidates via search across knowledge base

**Method 1.** Semantic search using **embeddings-based similarity**

Where is Cuddly?



[...] Huggy likes to work downstairs [...]

[...] Where is Paris located? [...]

[...] he was in a cuddly mood [...]

[ ...] Cuddly spends most days surrounded by books [...]

# Step 1: candidate retrieval

**Objective.** Select potentially-relevant candidates via search across knowledge base

**Method 2.** Search based on keywords matching via BM25



if we look for exact "word" containing  
in the document would be used  
"BM25" for keyword matching  
i.e. it is some function of overlap  
which gives heuristics score.

# Step 1: candidate retrieval

**Objective.** Select potentially-relevant candidates via search across knowledge base

**Method 2.** Search based on keywords matching via BM25

Where is Cuddly?

to find "cuddly" keyword  
in document / content  
BM25 helps to find out;  
Semantic Search won't  
help b/c Semantic  
embedding which does have  
some meaning but not specific keyword"

[...] Huggy likes to work downstairs [...]

[...] Where is Paris located? [...]

[...] he was in a cuddly mood [...]

[...] Cuddly spends most days surrounded by  
books [...]

# Step 1: candidate retrieval

**Objective.** Select potentially-relevant candidates via search across knowledge base

**Method 3.** Search based on hybrid combination of semantics and BM25



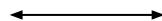
“Semantics + BM25”

# Step 1: candidate retrieval

**Objective.** Select potentially-relevant candidates via search across knowledge base

**Method 3.** Search based on **hybrid combination** of semantics and BM25

Where is Cuddly?



[...] Huggy likes to work downstairs [...]

[...] Where is Paris located? [...]

[...] he was in a cuddly mood [...]

[ ...] Cuddly spends most days surrounded by books [...]

# Extensions that can help with initial retrieval

- Mitigate discrepancy in nature of embeddings

↓  
when people wants to ask about something in LLM;  
but query is being added is in different nature  
w.r.t knowledge base (like short sentence, question)  
or many things

↓  
to resolve this; instead of computing Embedding related to prompt;  
first generate fake document based on prompt and then embed  
fake document to find relevant chunks.

# Extensions that can help with initial retrieval

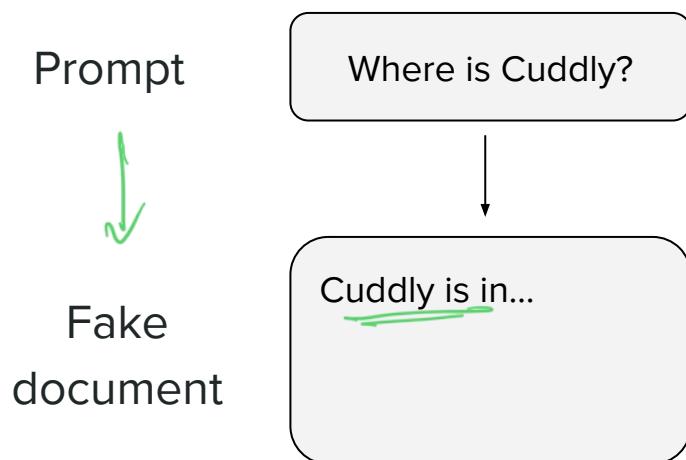
- Mitigate **discrepancy** in nature of embeddings

Prompt



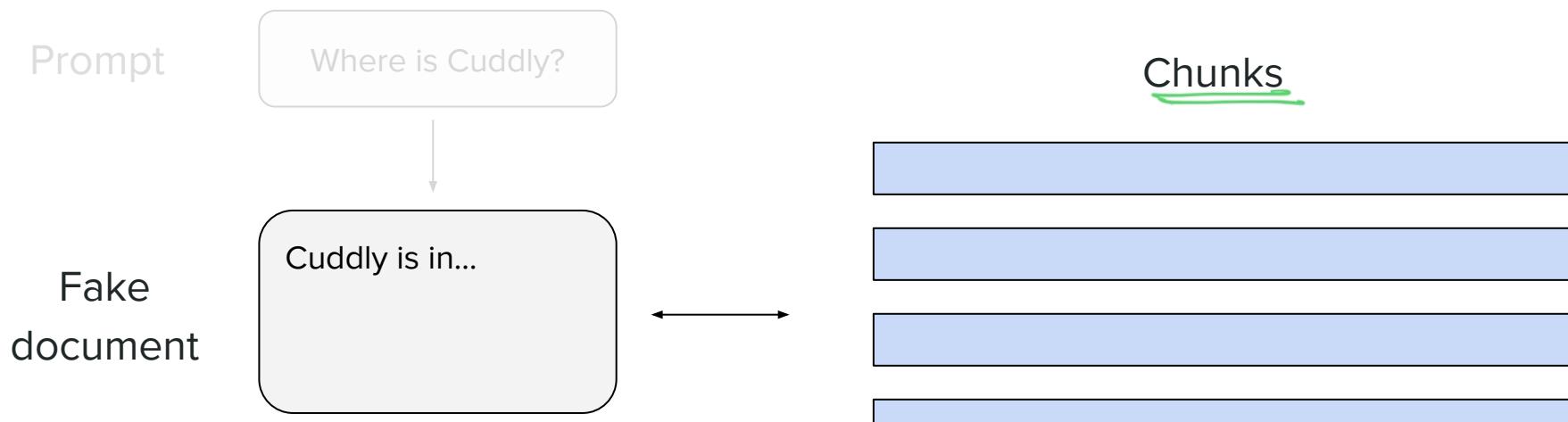
# Extensions that can help with initial retrieval

- Mitigate **discrepancy** in nature of embeddings



# Extensions that can help with initial retrieval

- Mitigate **discrepancy** in nature of embeddings



# Extensions that can help with initial retrieval

- Mitigate **discrepancy** in nature of embeddings
- **Contextualize** document chunks

↙  
how to make some of chunks?

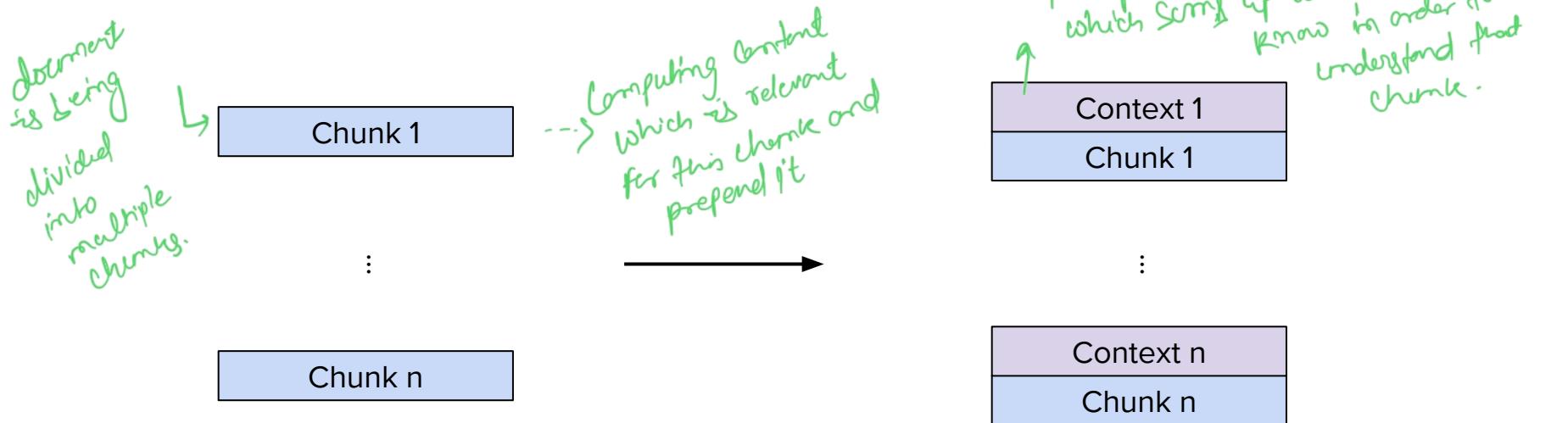
Chunk 1

:

Chunk n

# Extensions that can help with initial retrieval

- Mitigate **discrepancy** in nature of embeddings
- **Contextualize** document chunks



# Extensions that can help with initial retrieval

- Mitigate **discrepancy** in nature of embeddings
- **Contextualize** document chunks — it looks pricey, costly but to make it efficient  
i.e. generating content based on chunk and  
prepend it — Can be done using prompt  
Caching.

```
<document>
{WHOLE_DOCUMENT}
</document>
```

Here is the chunk we want to situate within the whole document:

```
{CHUNK_CONTENT}
```

Please give a short succinct context to situate this chunk within the overall document for the purposes of improving search retrieval of the chunk. Answer only with the succinct context and nothing else.

# Extensions that can help with initial retrieval

- Mitigate **discrepancy** in nature of embeddings
- **Contextualize** document chunks

**Prompt caching**

```
<document>
{WHOLE_DOCUMENT}
</document>
Here is the chunk we want to situate within the whole document:
{CHUNK_CONTENT}
```

Please give a short succinct context to situate this chunk within the overall document for the purposes of improving search retrieval of the chunk. Answer only with the succinct context and nothing else.

# Extensions that can help with initial retrieval

- Mitigate **discrepancy** in nature of embeddings
- **Contextualize** document chunks

## Pricing

 Copy page

### Text tokens

Prices per 1M tokens.

Batch   Flex   **Standard**   Priority

| MODEL             | INPUT  | CACHED INPUT | OUTPUT  |
|-------------------|--------|--------------|---------|
| gpt-5             | \$1.25 | \$0.125      | \$10.00 |
| gpt-5-mini        | \$0.25 | \$0.025      | \$2.00  |
| gpt-5-nano        | \$0.05 | \$0.005      | \$0.40  |
| gpt-5-chat-latest | \$1.25 | \$0.125      | \$10.00 |
| gpt-5-codex       | \$1.25 | \$0.125      | \$10.00 |

# Extensions that can help with initial retrieval

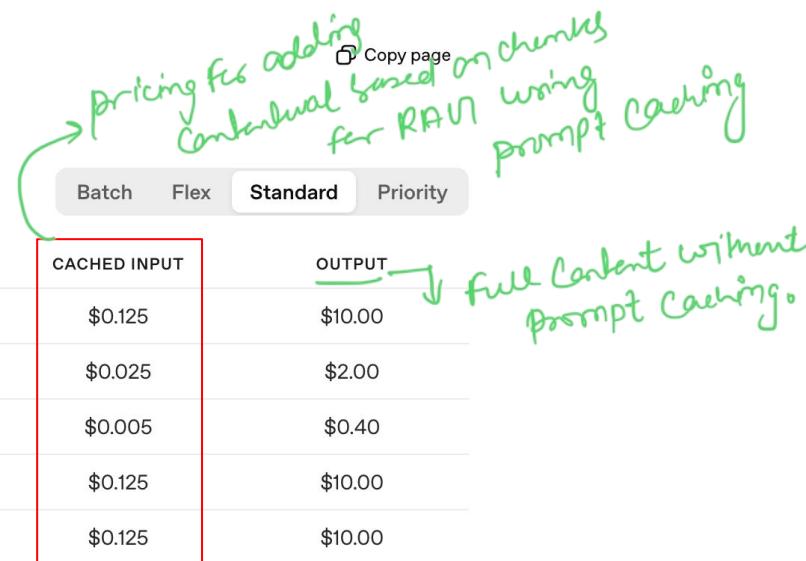
- Mitigate **discrepancy** in nature of embeddings
- **Contextualize** document chunks

## Pricing

### Text tokens

Prices per 1M tokens.

| MODEL             | INPUT  | CACHED INPUT | OUTPUT  |
|-------------------|--------|--------------|---------|
| gpt-5             | \$1.25 | \$0.125      | \$10.00 |
| gpt-5-mini        | \$0.25 | \$0.025      | \$2.00  |
| gpt-5-nano        | \$0.05 | \$0.005      | \$0.40  |
| gpt-5-chat-latest | \$1.25 | \$0.125      | \$10.00 |
| gpt-5-codex       | \$1.25 | \$0.125      | \$10.00 |



## Step 2: ranking

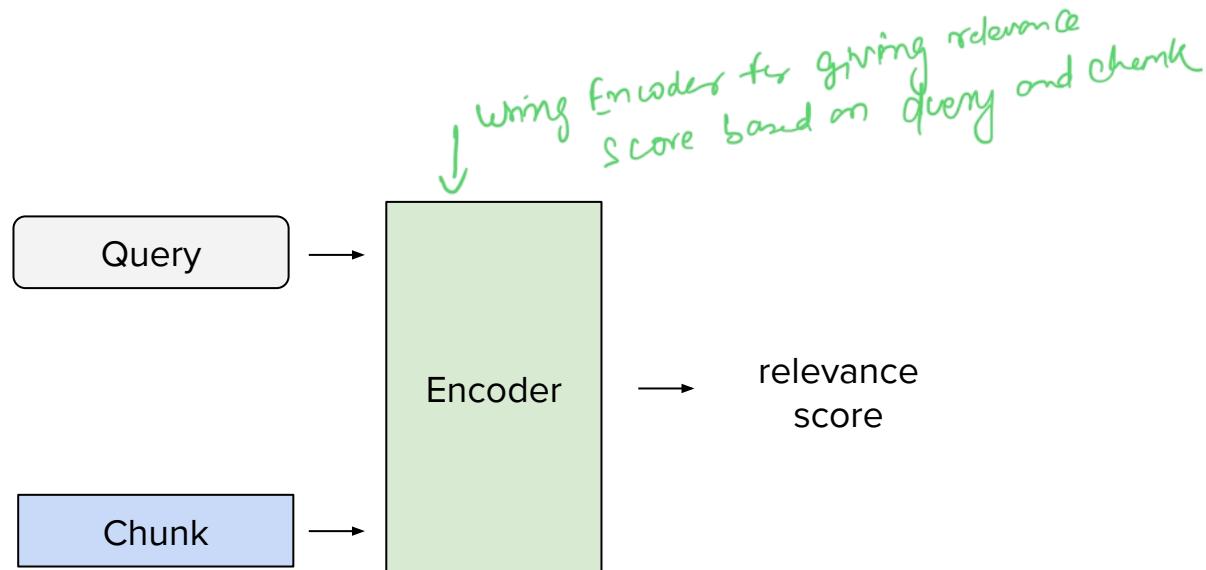
**Objective.** Provide final relevance score using more sophisticated (re-)ranker

Query

Chunk

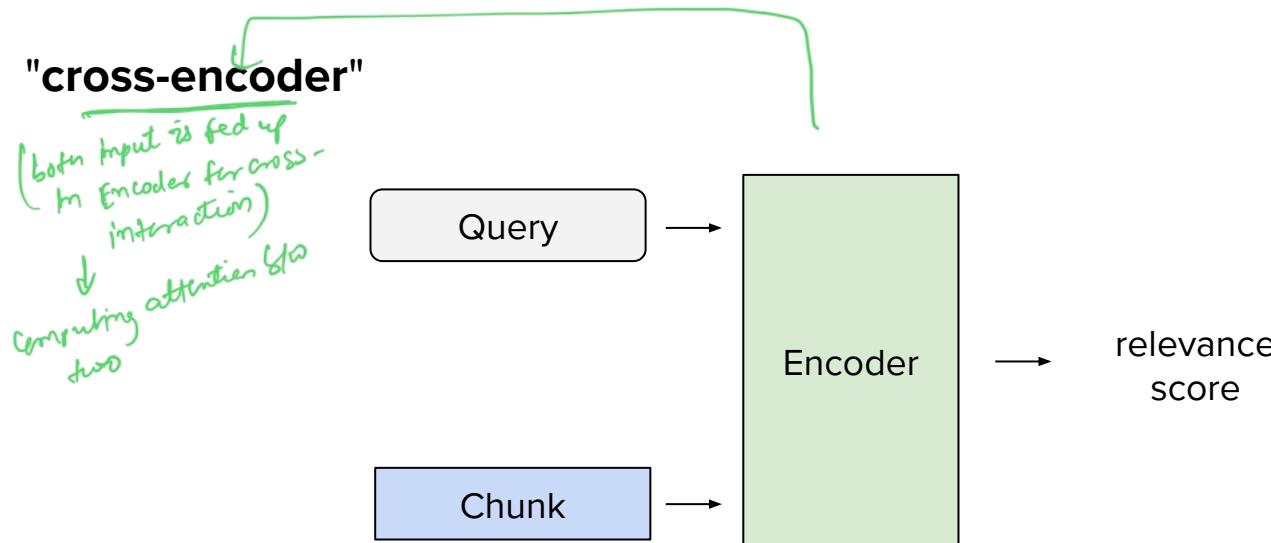
## Step 2: ranking

**Objective.** Provide final relevance score using more sophisticated (re-)ranker



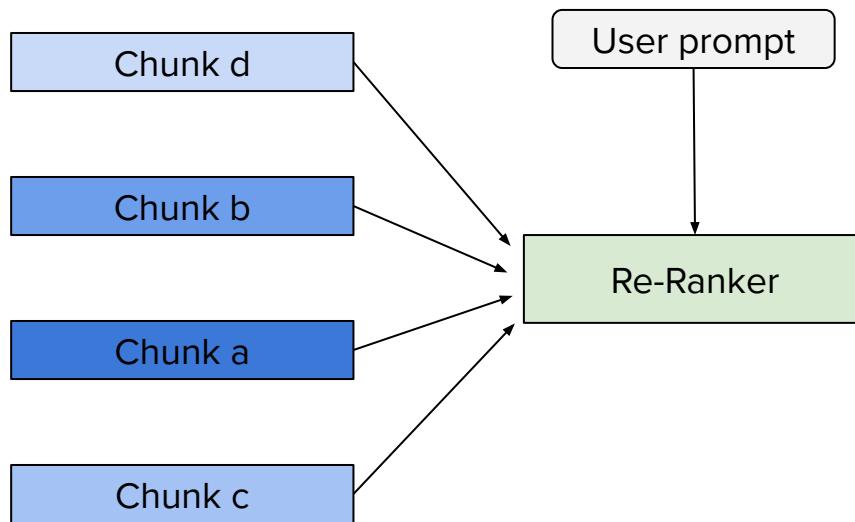
# Step 2: ranking

**Objective.** Provide final relevance score using more sophisticated (re-)ranger



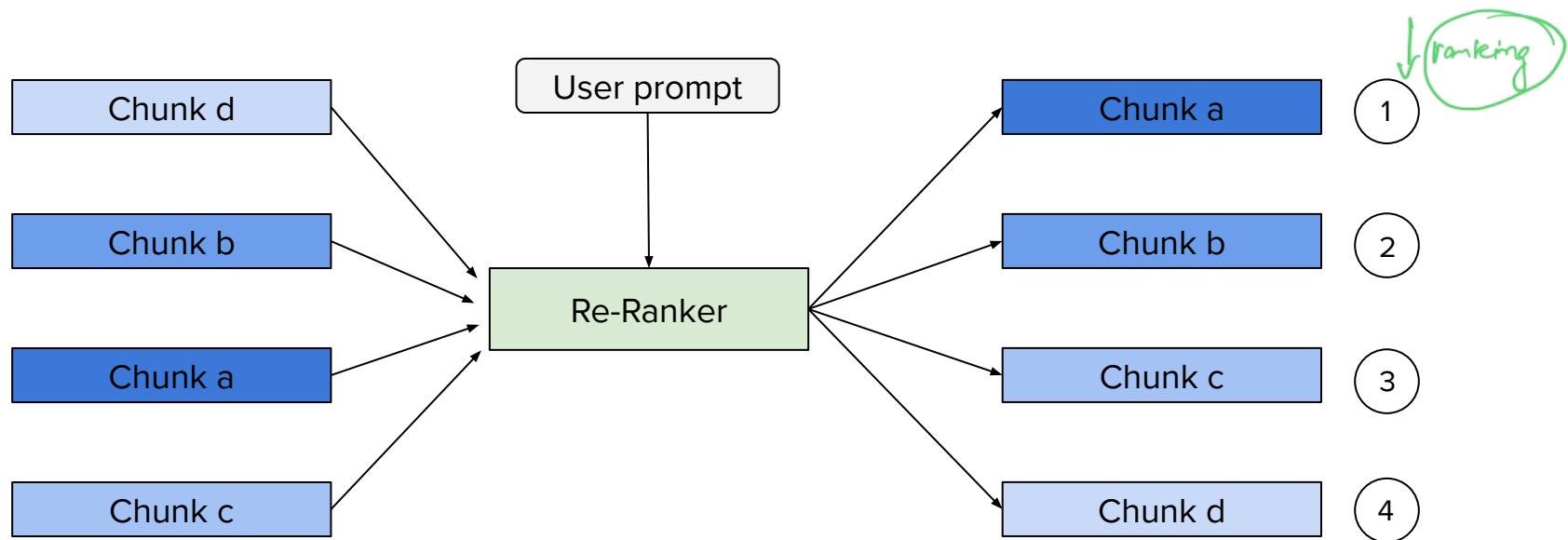
## Step 2: ranking

**Objective.** Provide final relevance score using more sophisticated (re-)ranker



# Step 2: ranking

**Objective.** Provide final relevance score using more sophisticated (re-)ranker



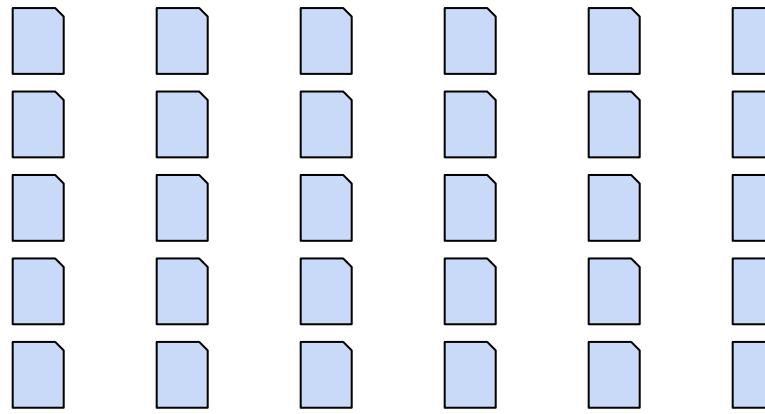
# Quantify performance of retrieval

**Setup.** Evaluate if **retrieved chunks** are **relevant**

You will have bunch of chunks  
do step 1 [finding relevant] and step 2 [ranking]  
chunks and find Top-K chunk relevant and compare  
b/w predicted Top-K chunks and actual relevant chunks  
once you have available n-chunks after ranking  
select only top-K  
for validation we use  
① NDCG (normalised discounted cumulative gain)  
② reciprocal rank  
③ recall@K  
④ precision@K.

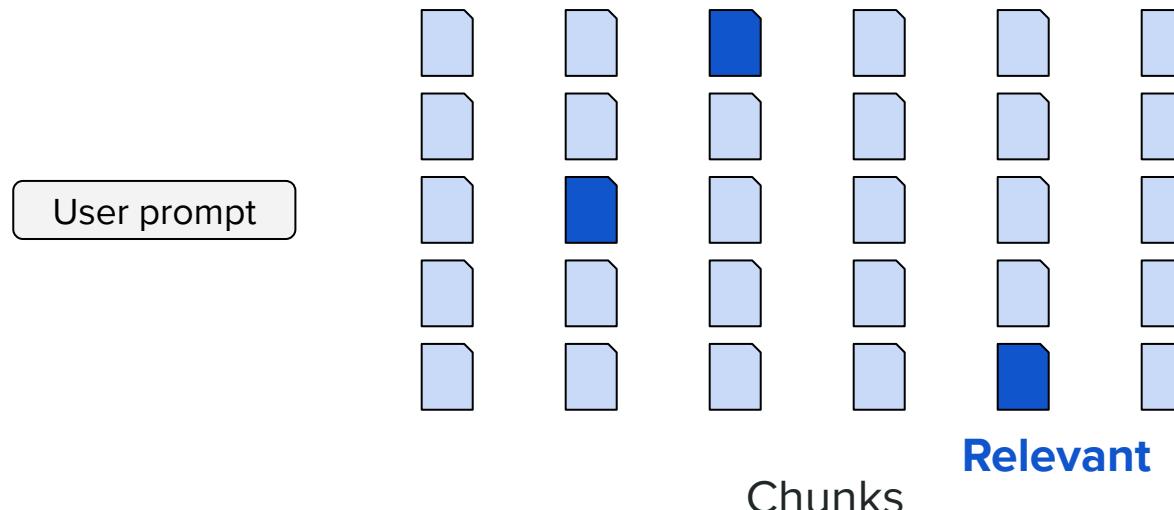
# Quantify performance of retrieval

**Setup.** Evaluate if **retrieved chunks** are **relevant**



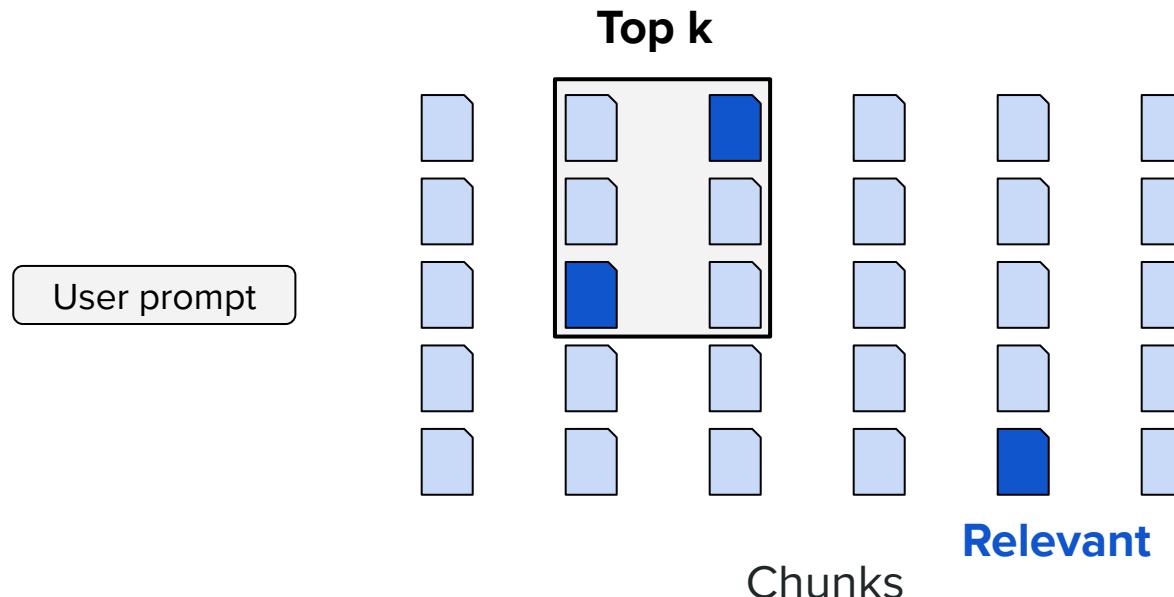
# Quantify performance of retrieval

**Setup.** Evaluate if **retrieved chunks** are **relevant**



# Quantify performance of retrieval

**Setup.** Evaluate if **retrieved chunks** are **relevant**



# Quantify performance of retrieval

**Setup.** Evaluate if **retrieved chunks** are **relevant**

## Ranking



⋮ ⋮



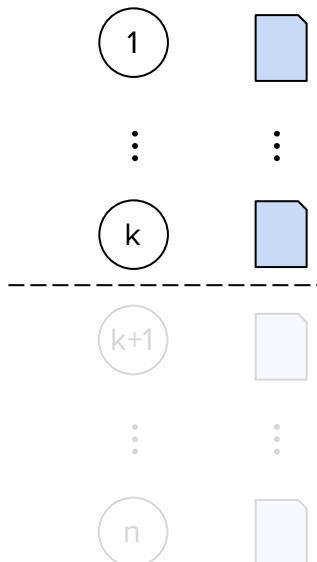
⋮ ⋮



# Quantify performance of retrieval

**Setup.** Evaluate if **retrieved chunks** are **relevant**

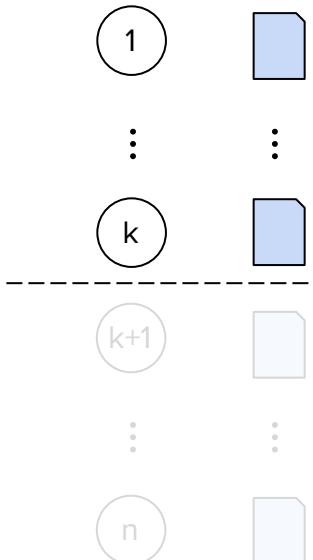
## Ranking



# Quantify performance of retrieval

**Setup.** Evaluate if **retrieved chunks** are **relevant**

## Ranking



- Normalized Discounted Cumulative Gain at k (**NDCG@k**)

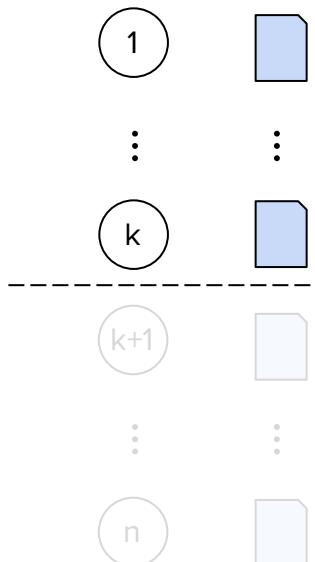
Tries to find how good your ranking is, by taking into consideration where you ranked relevant documents.

# Quantify performance of retrieval

**Setup.** Evaluate if **retrieved chunks** are **relevant**

*i.e. better to have most relevant document so in rank 1*

## Ranking



- Normalized Discounted Cumulative Gain at k (**NDCG@k**)

$$\text{DCG}@k = \sum_{i=1}^k \frac{\text{rel}_i}{\log_2(i + 1)}$$

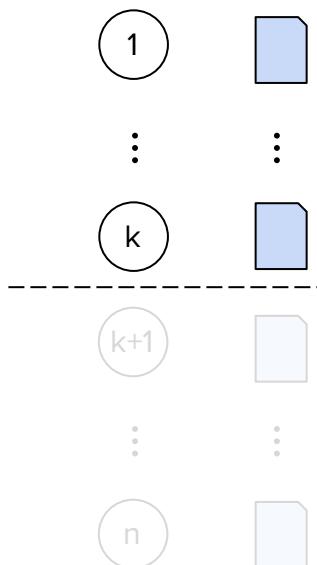
with  $\text{rel}_i \in \{0, 1\}$

*↑ DCG@k : if ranking relevant document is closer to gt.*

# Quantify performance of retrieval

**Setup.** Evaluate if **retrieved chunks** are **relevant**

## Ranking



- Normalized Discounted Cumulative Gain at  $k$  (**NDCG@k**)

$$\text{DCG}@k = \sum_{i=1}^k \frac{\text{rel}_i}{\log_2(i + 1)}$$

with  $\text{rel}_i \in \{0, 1\}$

$$\underline{\text{NDCG}@k} = \frac{\text{DCG}@k}{\overline{\text{IDCG}@k}}$$

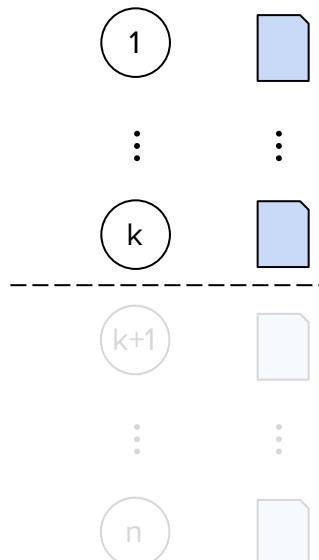
ideal/upper bound DC07

↑  
DCG@k if ranking was perfect

# Quantify performance of retrieval

**Setup.** Evaluate if **retrieved chunks** are **relevant**

## Ranking



- Normalized Discounted Cumulative Gain at  $k$  (NDCG@ $k$ )
- Reciprocal Rank at  $k$  (RR@ $k$ )

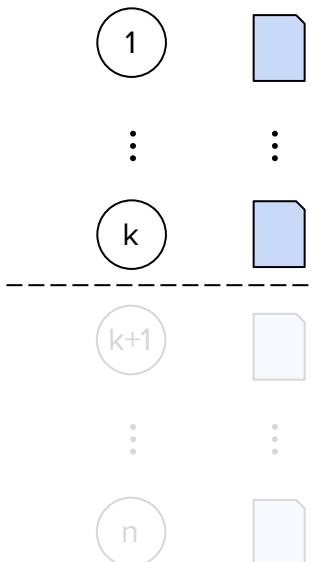
$$RR = \frac{1}{\text{rank}}$$

rank of the first relevant chunk

# Quantify performance of retrieval

**Setup.** Evaluate if **retrieved chunks** are **relevant**

## Ranking



- Normalized Discounted Cumulative Gain at  $k$  ( $\text{NDCG}@k$ )
- Reciprocal Rank at  $k$  ( $\text{RR}@k$ )
- **Recall at  $k$**

# Quantify performance of retrieval

**Setup.** Evaluate if **retrieved chunks** are **relevant**

## Ranking



- Normalized Discounted Cumulative Gain at k (**NDCG@k**)
- Reciprocal Rank at k (**RR@k**)
- **Recall at k**



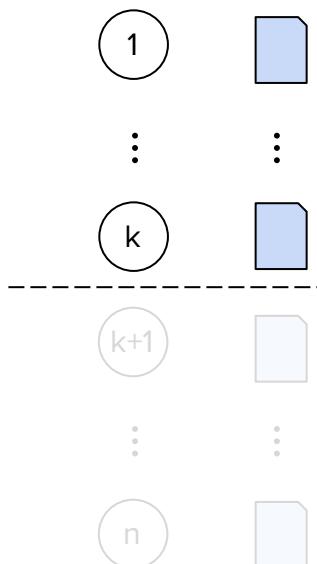
$$\text{Recall}@k = \frac{|\text{relevant in top } k|}{|\text{relevant}|}$$

*↑  
the class of relevancy*

# Quantify performance of retrieval

**Setup.** Evaluate if **retrieved chunks** are **relevant**

## Ranking

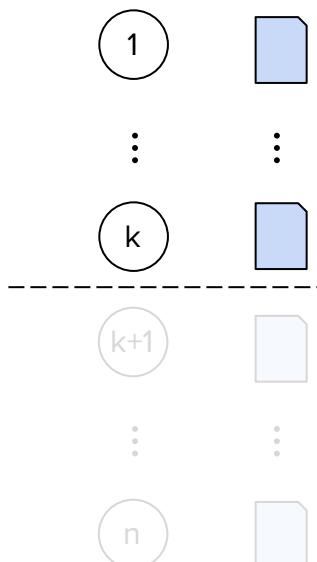


- Normalized Discounted Cumulative Gain at  $k$  (**NDCG@k**)
- Reciprocal Rank at  $k$  (**RR@k**)
- Recall at  $k$
- **Precision at  $k$**

# Quantify performance of retrieval

**Setup.** Evaluate if **retrieved chunks** are **relevant**

## Ranking



- Normalized Discounted Cumulative Gain at  $k$  (**NDCG@k**)
- Reciprocal Rank at  $k$  (**RR@k**)
- Recall at  $k$
- **Precision at  $k$**

$$\text{Precision}@k = \frac{\text{relevant in top } k}{k}$$



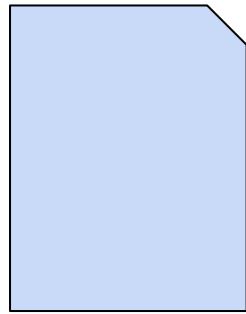
# Transformers & Large Language Models

RAG

Tool calling

Agents

# Motivation



**Unstructured**

RAG

# Motivation



Unstructured  
RAG

unstructured data  
to structured  
format for  
writing tools  
“RAG”

| ID  | Field | ... |
|-----|-------|-----|
| 123 | Obs   | ... |
| ... | ...   | ... |

Structured

# Idea

| ID  | Field | ... |
|-----|-------|-----|
| 123 | Obs   | ... |
| ... | ...   | ... |

"tool-calling"  
relationship b/w  
structured data  
with API

## Function

```
def get_data(
 id, field, ...
):
 # Logic.
 ...
 return result
```

## Definition

**"Tool calling** [...] allows autonomous systems to complete complex tasks by dynamically accessing and [may act] upon external resources."

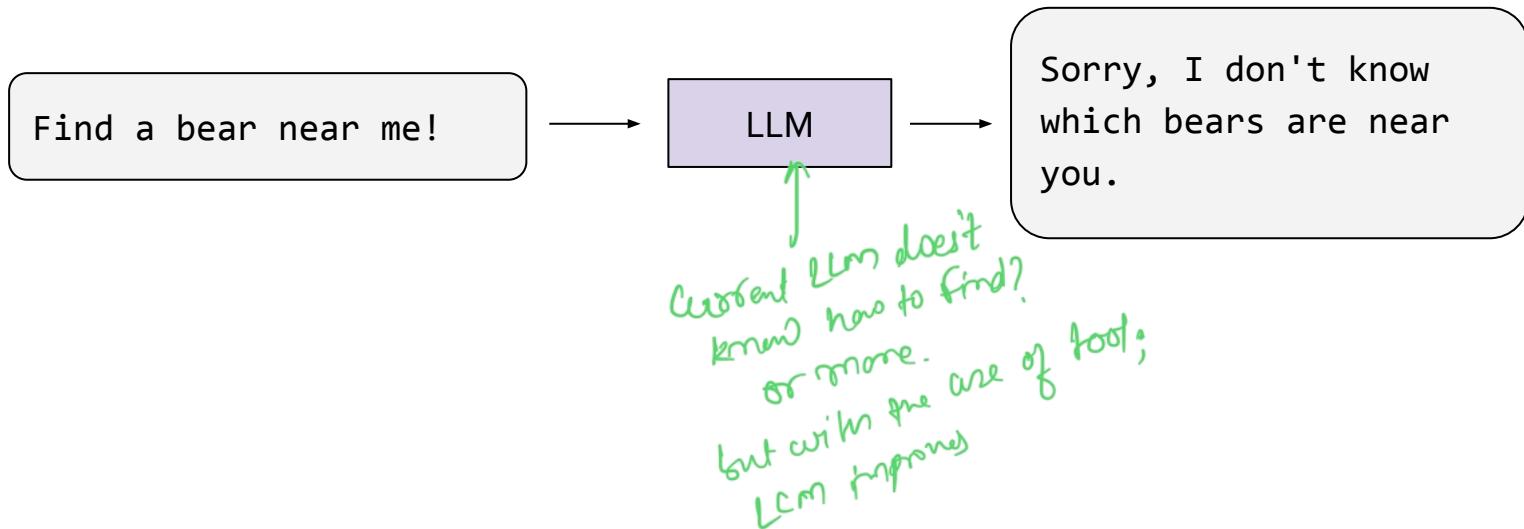
# Definition

"Tool calling [...] allows autonomous systems to **complete** complex **tasks** by dynamically accessing and [may act] upon **external resources**."

*↓*  
motion of completing tasks  
but reliance on external resources.  
(basically filling the knowledge gap  
LLM does have

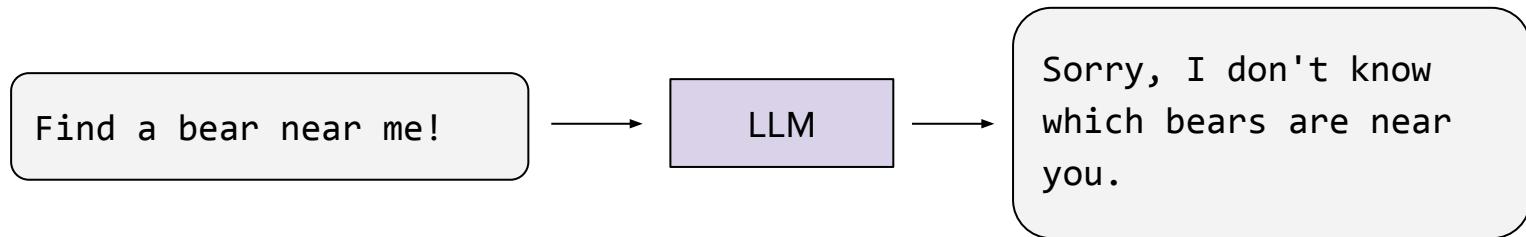
# Real-life example

LLM as we've known it so far

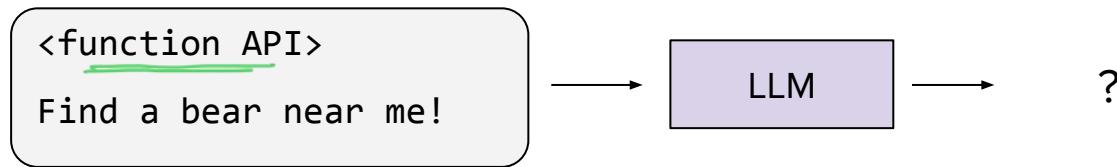


# Real-life example

LLM as we've known it so far

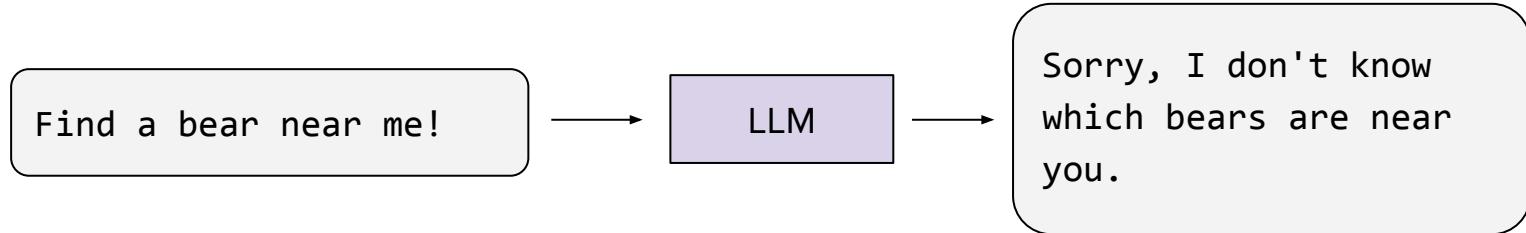


LLM with tools

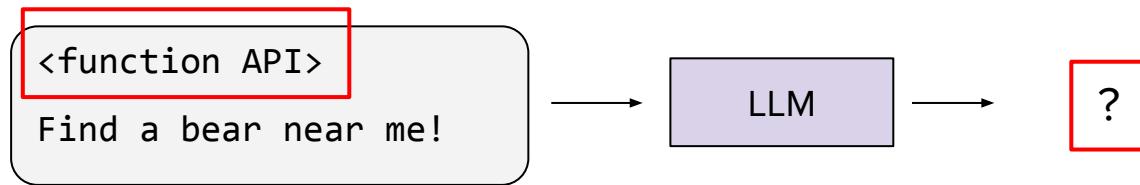


# Real-life example

LLM as we've known it so far



LLM with tools



# Real-life example

## find\_teddy\_bear.py

```
from dataclasses import dataclass
from geopy.distance import geodesic
import requests

@dataclass
class TeddyBearInfo:
 name: str
 distance_meters: float
 mood: str
 message: str

def find_teddy_bear(location: tuple[float, float]) -> TeddyBearInfo:
 """
 Finds the nearest teddy bear to the given GPS coordinates.

 Parameters:
 location: A (latitude, longitude) pair representing the user's current
 location.

 Returns:
 TeddyBearInfo: Information about the nearest teddy bear found.
 """
 # Call API to get the closest teddy bear
 user_lat, user_lon = location
 api_url = "https://api.to.teddy.bears.com/v1/closest"
 try:
 response = requests.get(
 api_url,
 params={"latitude": user_lat, "longitude": user_lon},
 timeout=5
)
 response.raise_for_status()
 closest_teddy_bear = response.json()
 except requests.RequestException as e:
 raise RuntimeError(f"Failed to fetch teddy bear data from API: {e}")
```

```
Extract coordinates from API response
bear_lat, bear_lon = closest_teddy_bear["coords"]

Compute distance to the bear using geopy (returns distance in meters)
distance = geodesic((user_lat, user_lon), (bear_lat, bear_lon)).meters

return TeddyBearInfo(
 name=closest_teddy_bear["name"],
 distance_meters=round(distance, 2),
 mood=closest_teddy_bear["mood"],
 message=f"{closest_teddy_bear['name']} is {closest_teddy_bear['mood']} "
 f"and only {round(distance, 2)} meters away!"
```

location might be written  
in form API writes I'm will  
return later

# Real-life example

## find\_teddy\_bear.py

```
from dataclasses import dataclass
from geopy.distance import geodesic
import requests

@dataclass
class TeddyBearInfo:
 name: str
 distance_meters: float
 mood: str
 message: str

def find_teddy_bear(location: tuple[float, float]) -> TeddyBearInfo:
 """
 Finds the nearest teddy bear to the given GPS coordinates.

 Parameters:
 location: A (latitude, longitude) pair representing the user's current
 location.

 Returns:
 TeddyBearInfo: Information about the nearest teddy bear found.
 """
 # Call API to get the closest teddy bear
 user_lat, user_lon = location
 api_url = "https://api.to.teddy.bears.com/v1/closest"
 try:
 response = requests.get(
 api_url,
 params={"latitude": user_lat, "longitude": user_lon},
 timeout=5
)
 response.raise_for_status()
 closest_teddy_bear = response.json()
 except requests.RequestException as e:
 raise RuntimeError(f"Failed to fetch teddy bear data from API: {e}")
```

```
Extract coordinates from API response
bear_lat, bear_lon = closest_teddy_bear["coords"]

Compute distance to the bear using geopy (returns distance in meters)
distance = geodesic((user_lat, user_lon), (bear_lat, bear_lon)).meters

return TeddyBearInfo(
 name=closest_teddy_bear["name"],
 distance_meters=round(distance, 2),
 mood=closest_teddy_bear["mood"],
 message=f"{closest_teddy_bear['name']} is {closest_teddy_bear['mood']} "
 f"and only {round(distance, 2)} meters away!"
```

has a descriptive,  
well-documented API

# Real-life example

## find\_teddy\_bear.py

```
from dataclasses import dataclass
from geopy.distance import geodesic
import requests

@dataclass
class TeddyBearInfo:
 name: str
 distance_meters: float
 mood: str
 message: str

def find_teddy_bear(location: tuple[float, float]) -> TeddyBearInfo:
 """
 Finds the nearest teddy bear to the given GPS coordinates.

 Parameters:
 location: A (latitude, longitude) pair representing the user's current
 location.

 Returns:
 TeddyBearInfo: Information about the nearest teddy bear found.
 """
 # Call API to get the closest teddy bear
 user_lat, user_lon = location
 api_url = "https://api.to.teddy.bears.com/v1/closest"
 try:
 response = requests.get(
 api_url,
 params={"latitude": user_lat, "longitude": user_lon},
 timeout=5
)
 response.raise_for_status()
 closest_teddy_bear = response.json()
 except requests.RequestException as e:
 raise RuntimeError(f"Failed to fetch teddy bear data from API: {e}")
```

```
Extract coordinates from API response
bear_lat, bear_lon = closest_teddy_bear["coords"]

Compute distance to the bear using geopy (returns distance in meters)
distance = geodesic((user_lat, user_lon), (bear_lat, bear_lon)).meters

return TeddyBearInfo(
 name=closest_teddy_bear["name"],
 distance_meters=round(distance, 2),
 mood=closest_teddy_bear["mood"],
 message=f"{closest_teddy_bear['name']} is {closest_teddy_bear['mood']} "
 f"and only {round(distance, 2)} meters away!"
```

(optional) has some  
backend call

# Real-life example

## find\_teddy\_bear.py

```
from dataclasses import dataclass
from geopy.distance import geodesic
import requests

@dataclass
class TeddyBearInfo:
 name: str
 distance_meters: float
 mood: str
 message: str

def find_teddy_bear(location: tuple[float, float]) -> TeddyBearInfo:
 """
 Finds the nearest teddy bear to the given GPS coordinates.

 Parameters:
 location: A (latitude, longitude) pair representing the user's current
 location.

 Returns:
 TeddyBearInfo: Information about the nearest teddy bear found.
 """
 # Call API to get the closest teddy bear
 user_lat, user_lon = location
 api_url = "https://api.to.teddy.bears.com/v1/closest"
 try:
 response = requests.get(
 api_url,
 params={"latitude": user_lat, "longitude": user_lon},
 timeout=5
)
 response.raise_for_status()
 closest_teddy_bear = response.json()
 except requests.RequestException as e:
 raise RuntimeError(f"Failed to fetch teddy bear data from API: {e}")

 return TeddyBearInfo(
 name=closest_teddy_bear["name"],
 distance_meters=round(distance, 2),
 mood=closest_teddy_bear["mood"],
 message=f"{closest_teddy_bear['name']} is {closest_teddy_bear['mood']} "
 f"and only {round(distance, 2)} meters away!"
)
```

```
Extract coordinates from API response
bear_lat, bear_lon = closest_teddy_bear["coords"]

Compute distance to the bear using geopy (returns distance in meters)
distance = geodesic((user_lat, user_lon), (bear_lat, bear_lon)).meters

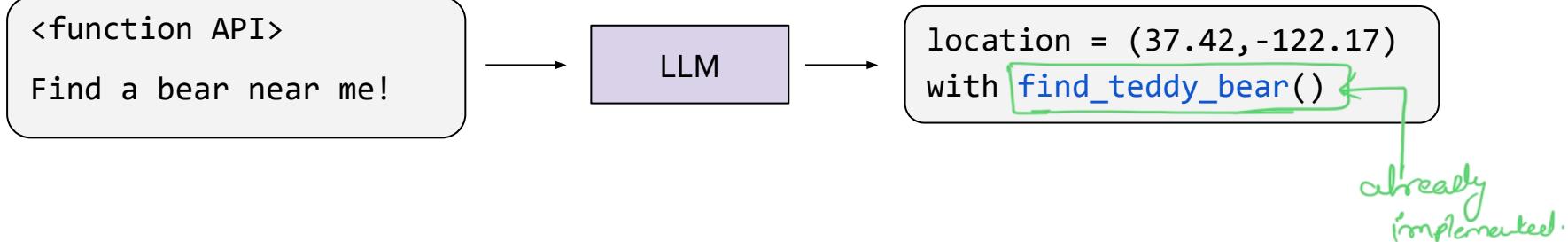
return TeddyBearInfo(
 name=closest_teddy_bear["name"],
 distance_meters=round(distance, 2),
 mood=closest_teddy_bear["mood"],
 message=f"{closest_teddy_bear['name']} is {closest_teddy_bear['mood']} "
 f"and only {round(distance, 2)} meters away!"
```

returns some info

# How it works

# How it works

1. Let LLM find **argument** for **relevant function call**

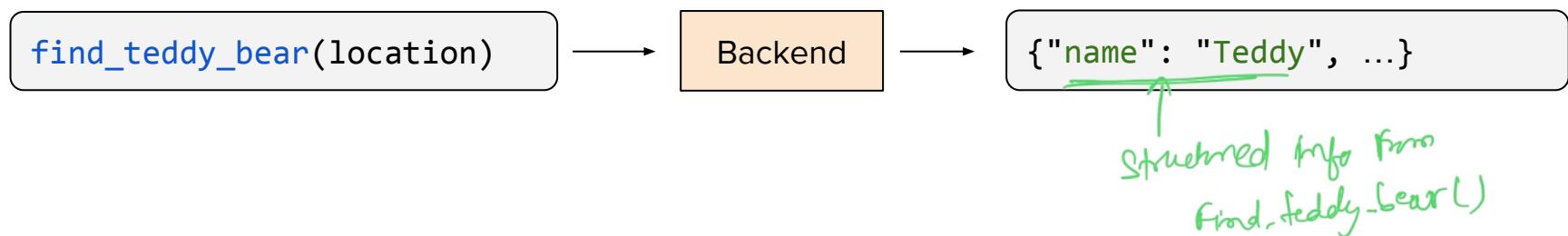


# How it works

1. Let **LLM** find **argument** for **relevant function call**



2. Make **function call**



# How it works

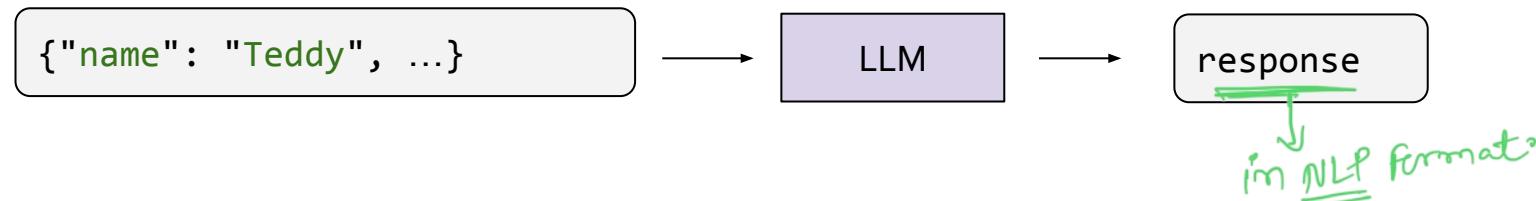
- ✓ 1. Let LLM find argument for relevant function call



- ✓ 2. Make function call



- ✓ 3. Let LLM deduce conclusion based on results



# Teach a model to use a tool

**Method 1:** via training

# Teach a model to use a tool

## Method 1: via training

```
<function API>
Find a bear near me!
```

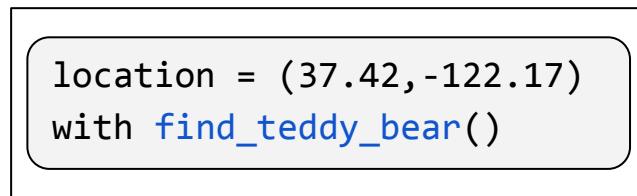


```
location = (37.42, -122.17)
with find_teddy_bear()
```

Tool prediction

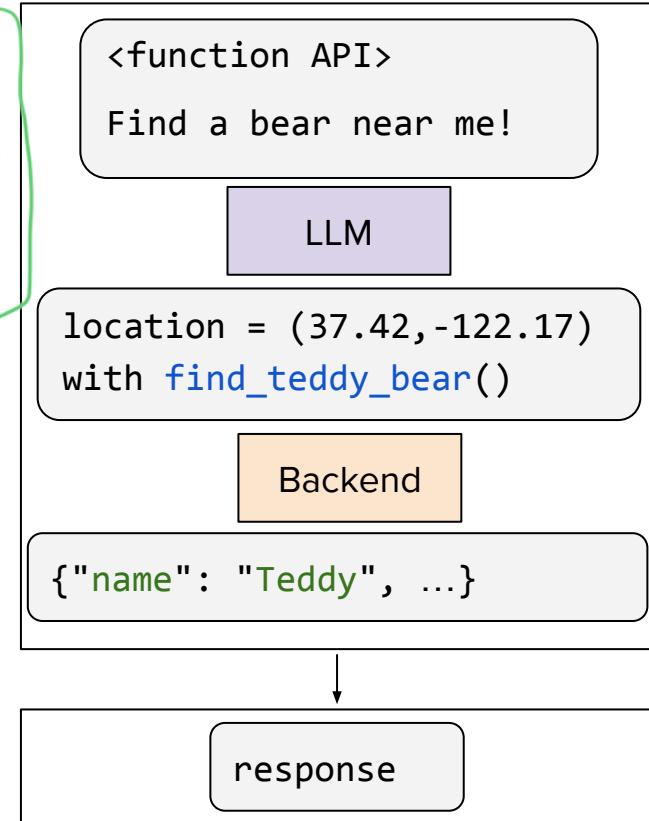
# Teach a model to use a tool

**Method 1:** via training



Tool prediction

~~SFT~~ pair for training  
tool to work  
or take action  
based on  
tool-calling



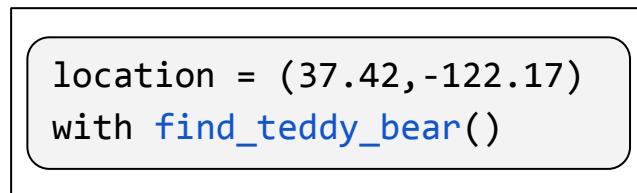
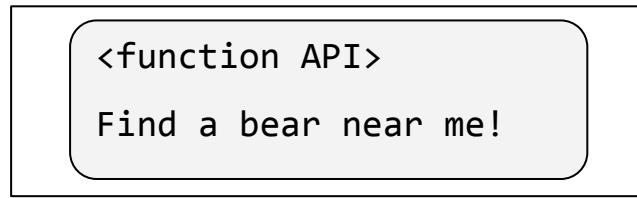
Response generation

# Teach a model to use a tool

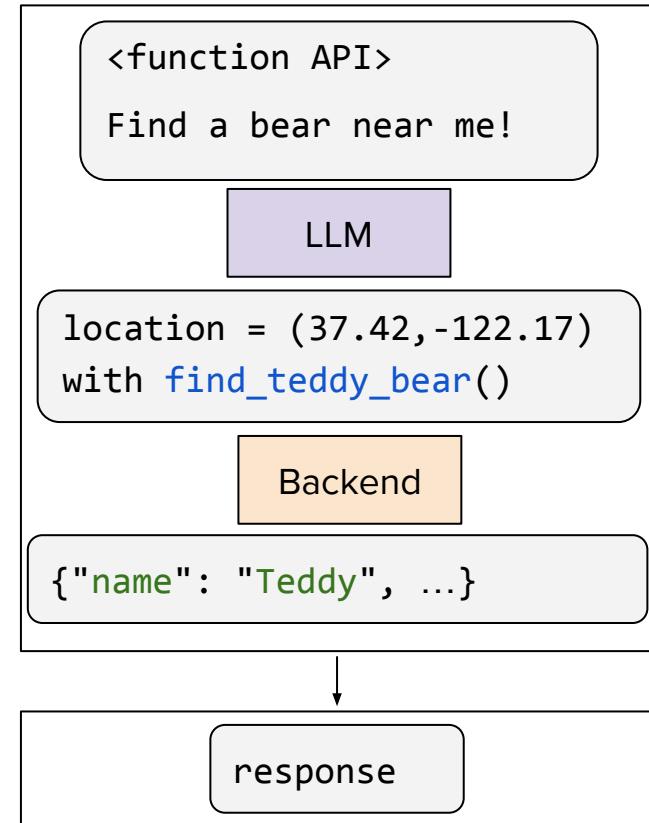
**Method 1:** via training

[SFT pair generated for tool)

conversation  
history so far



Tool prediction

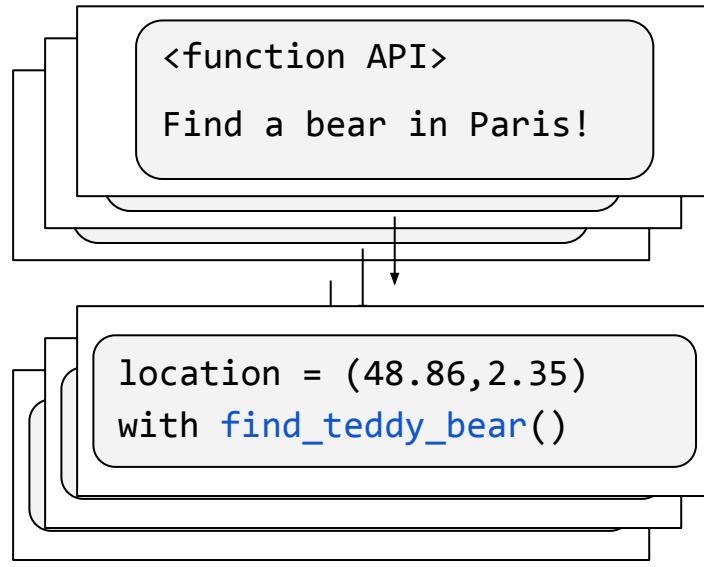


desired  
prediction

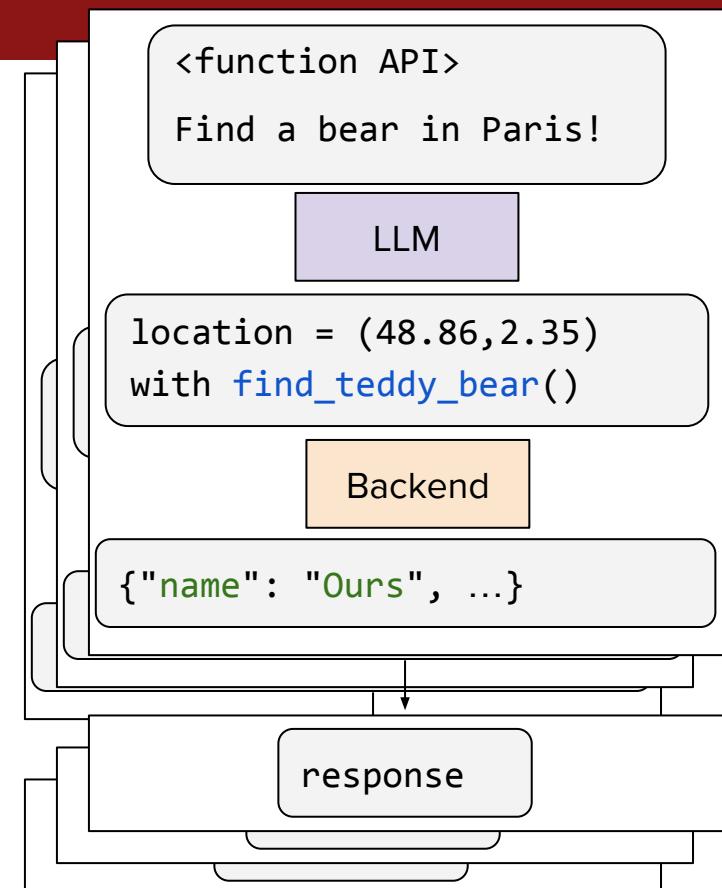
Response generation

# Teach a model to use a tool

## Method 1: via training



Tool prediction



Response generation

# Teach a model to use a tool

Method 1: via training

Method 2: via prompting

// LLMs are so smart nowadays  
so, using prompting can also  
be with instruction.  
(without training)

<function API> + <detailed explanation on how to use it>

Find a bear near me!

# Teach a model to use a tool

Method 1: via training

Method 2: via prompting

how to write such a  
description?

(using SFT training or  
reasoning model)

```
<function API> + <detailed explanation on how to use it>
```

Find a bear near me!

# Teach a model to use a tool

Method 1: via training

Method 2: via prompting

how to write such a  
description?

```
<function API> + <detailed explanation on how to use it>
```

Find a bear near me!

(recommended)

One way:

Use SFT pairs as evaluation + use a powerful reasoning model  
to write it for you!

# Examples of common use cases

## Information

- ✓ Web/database search
- ✓ Weather, stocks, and any other tracker
- ✓ Codebase

## Computation

- ✓ Calculator
  - ✓ Code execution (often in Python)
- ...and many more!

## Action

- ✓ Send emails/messages and other in-computer action
- ✓ anything else within the domain of an assistant

# In practice: many tools



for many action using LLM tooling

```
def find_teddy_bear(
 location: tuple[float,
 float]
) -> TeddyBearInfo:
 # ...
```

```
def hug_teddy_bear(
 recipient: TeddyBear,
 intensity: str = "warm"
) -> HugResponse:
 # ...
```

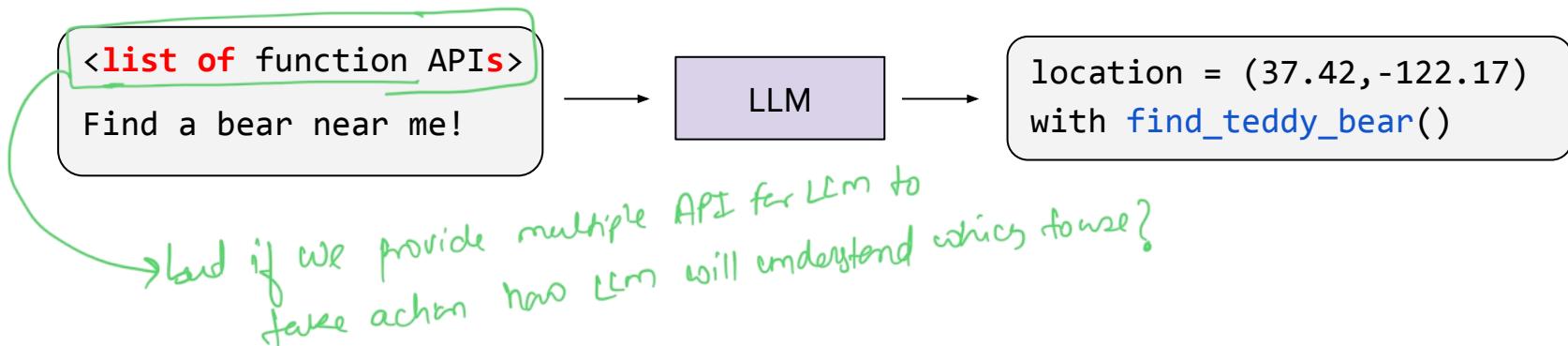
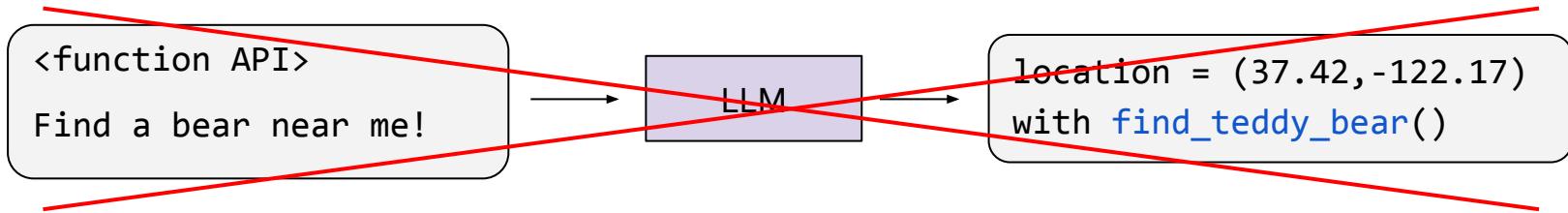
```
def check_teddy_mood(
 name: TeddyBear
) -> TeddyMood:
 # ...
```

```
def send_teddy_gift(
 recipient: TeddyBear,
 gift: str = "poetry book"
) -> GiftResponse:
 # ...
```

```
def schedule_playdate(
 host: TeddyBear,
 guest: TeddyBear,
 time: datetime
) -> Confirmation:
 # ...
```

```
def send_message(
 recipient: TeddyBear,
 message: str
) -> MessageResponse:
 # ...
```

# In practice: many tools



# Tools summary

## Benefits.

- LLMs just became way more useful!
- They can also interact with the real world
- Overcomes "knowledge cutoff" limitation

↓  
RAM (adding more relevant  
information  
after cut off date)

# Tools summary

## Benefits.

- LLMs just became way more useful!
- They can also interact with the real world
- Overcomes "knowledge cutoff" limitation

## Challenges.

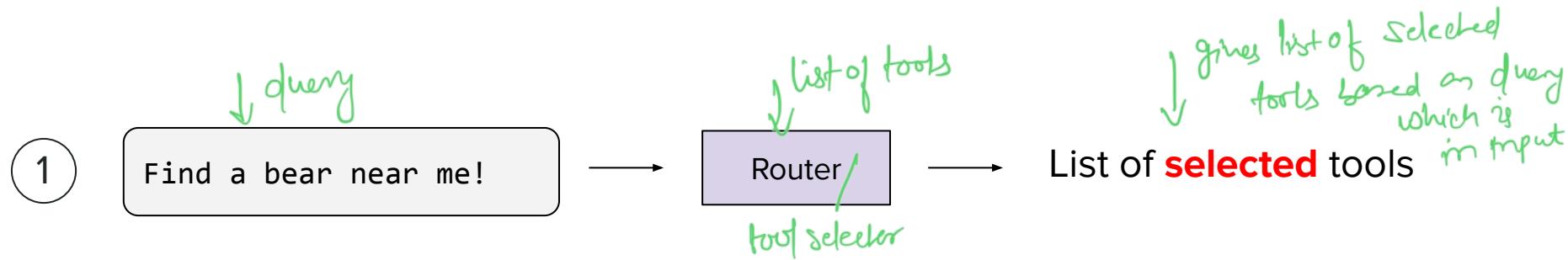
- More tools = decrease performance
- Finite context length: not scalable
- Many tools to define. Lots of work.

# Tool selection

**Goal.** Both reduce latency and improve performance

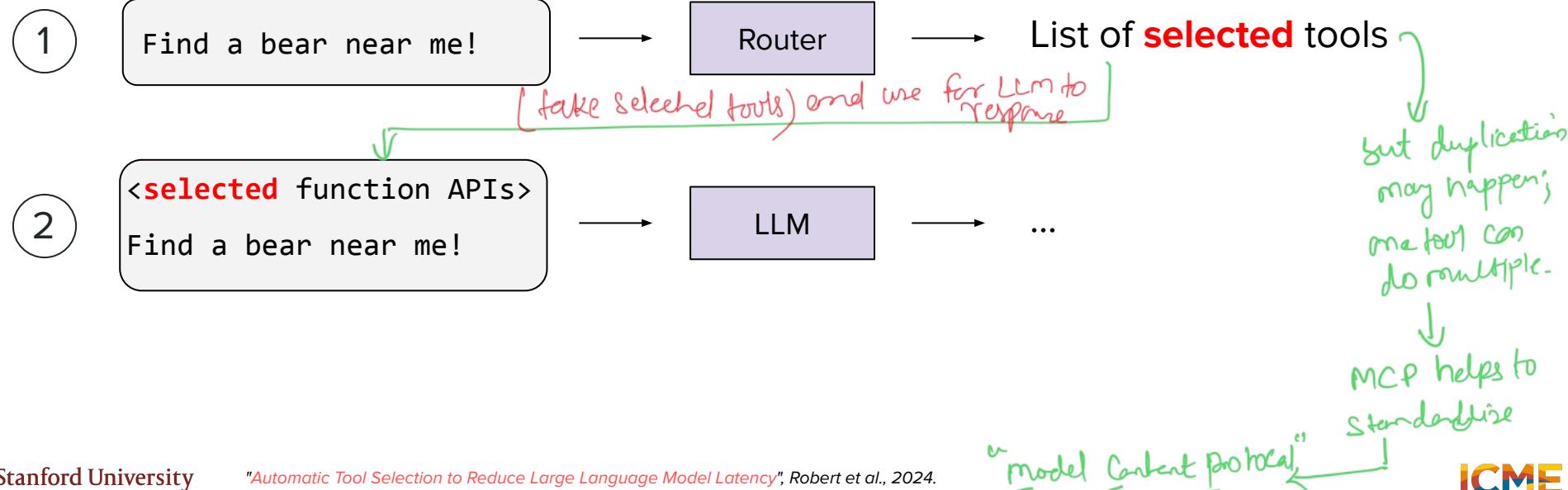
# Tool selection

**Goal.** Both reduce latency and improve performance



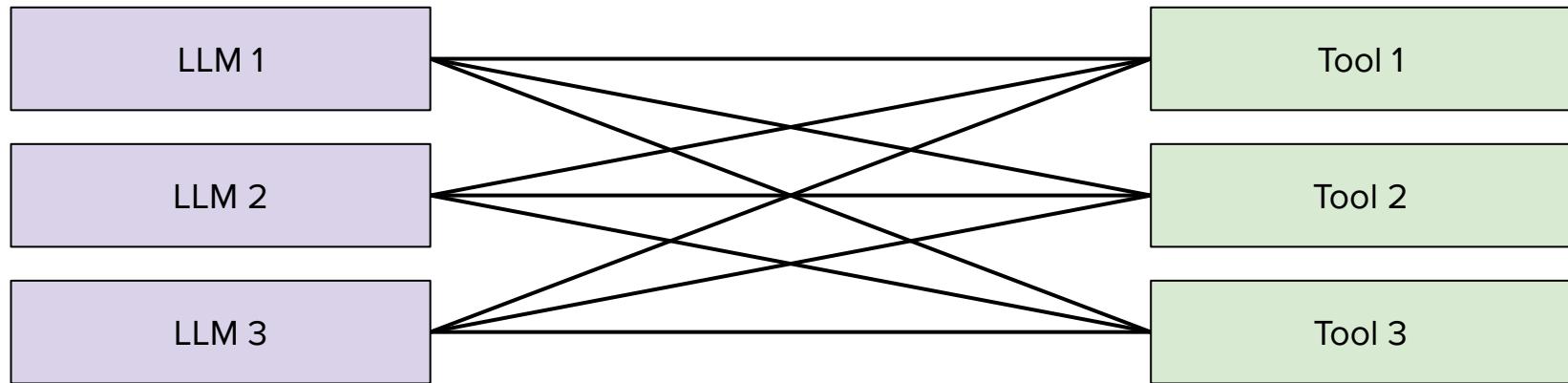
# Tool selection

**Goal.** Both reduce latency and improve performance



# Motivation for standardized protocol

**Goal.** Avoid duplication of tool implementations



# Standardization: MCP

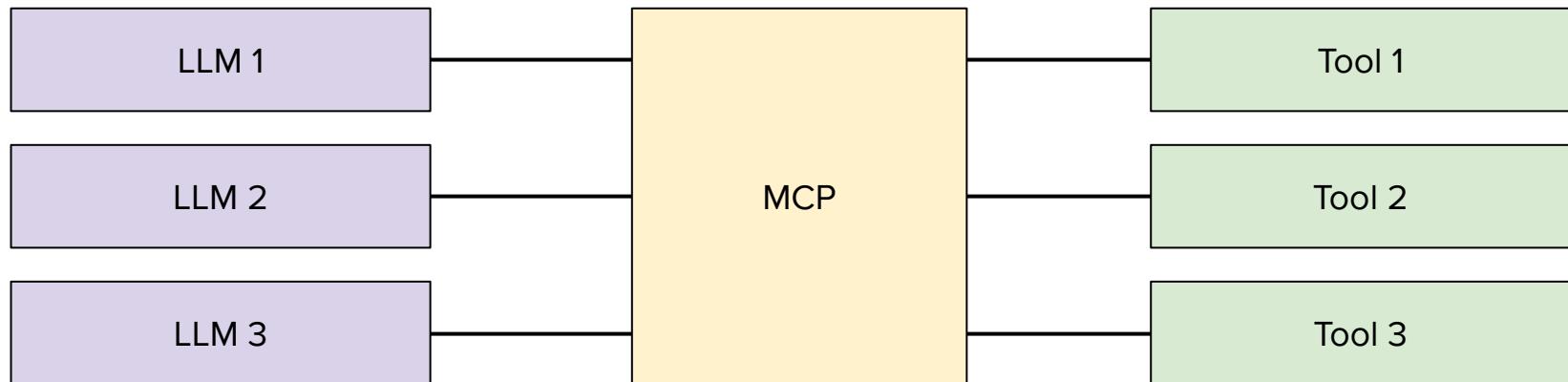
**MCP = Model Context Protocol**

**Idea.** Connect tools/data to LLMs in a standard way

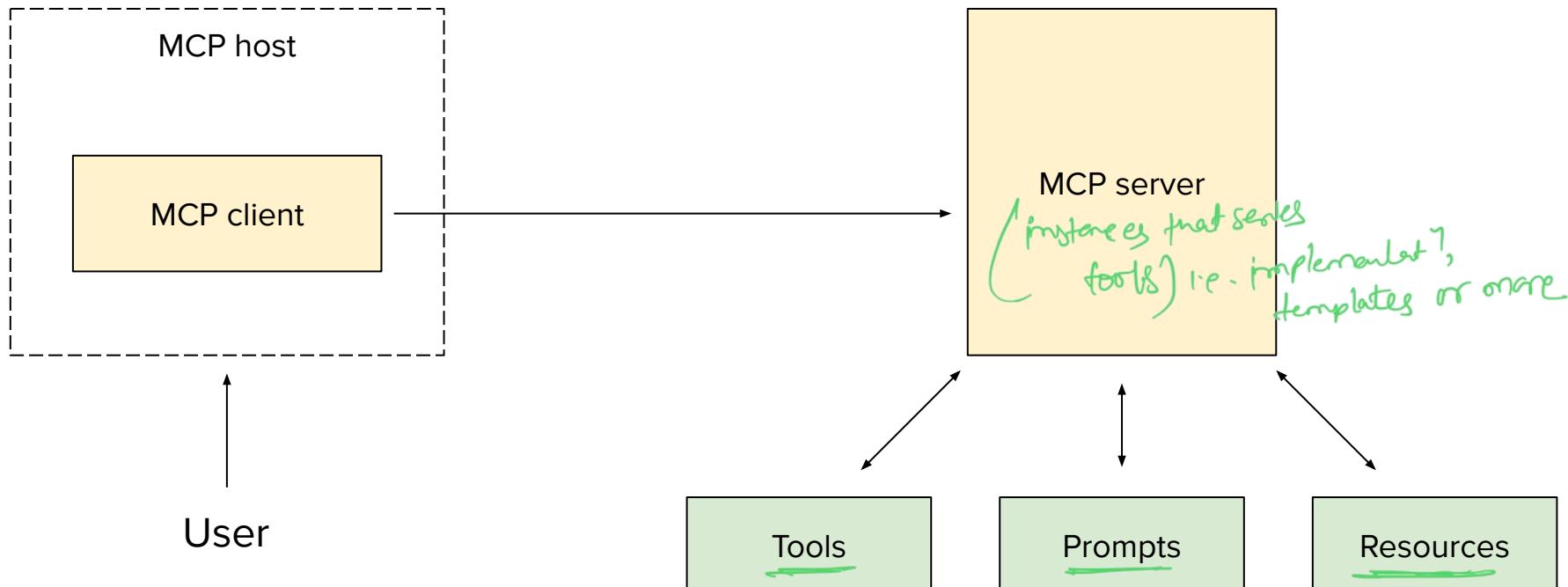
# Standardization: MCP

**MCP = Model Context Protocol**

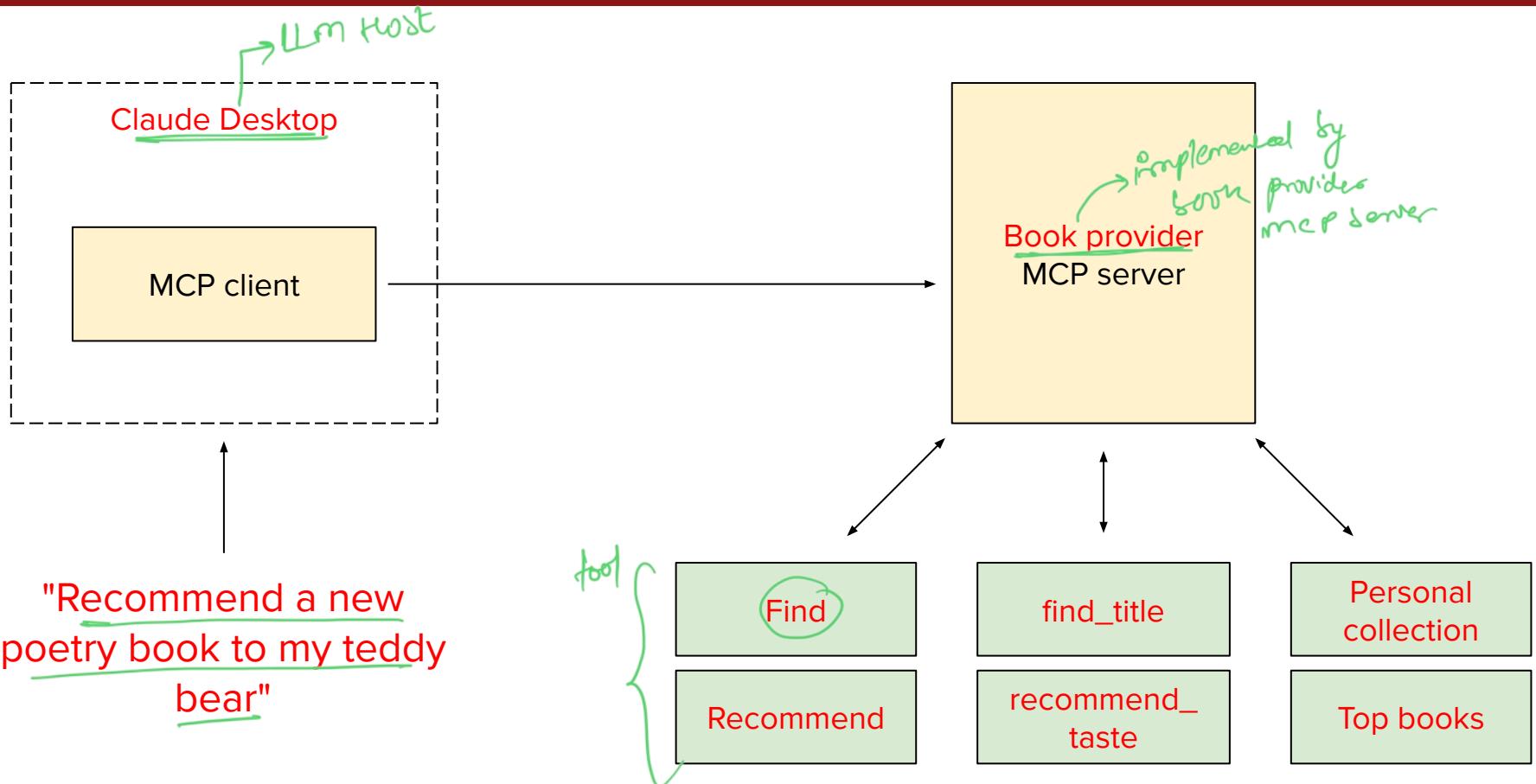
**Idea.** Connect tools/data to LLMs in a standard way



# Standardization: MCP



# Standardization: MCP





# Transformers & Large Language Models

RAG

Tool calling

Agents



# Definition

"An **agent** is a system that autonomously

**pursues goals** and **completes tasks on a**

**user's behalf.**"

# Definition

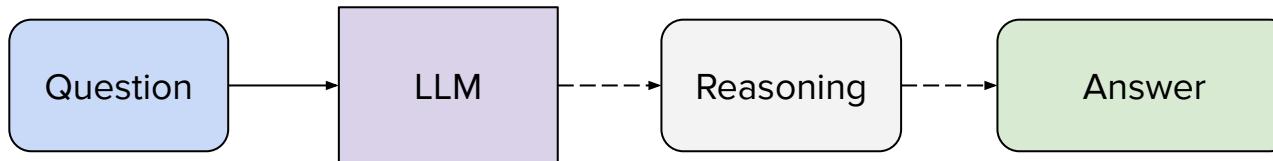
"An **agent** is a system that **autonomously**  
**pursues goals** and **completes tasks** on a  
user's behalf."

# High-level idea

## Traditional

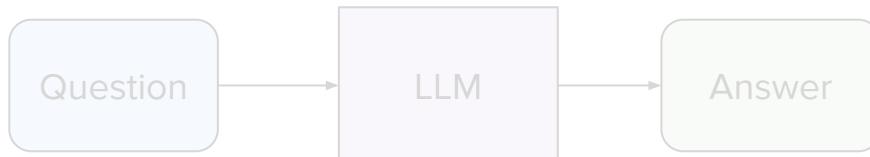


## Reasoning

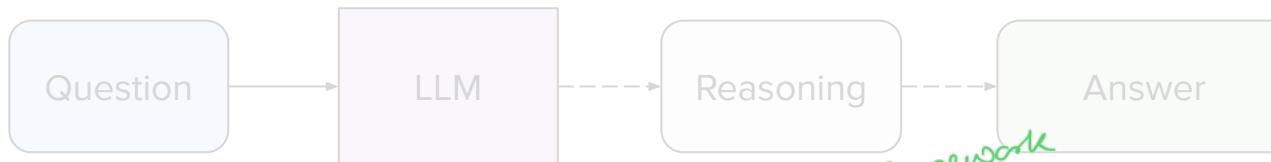


# High-level idea

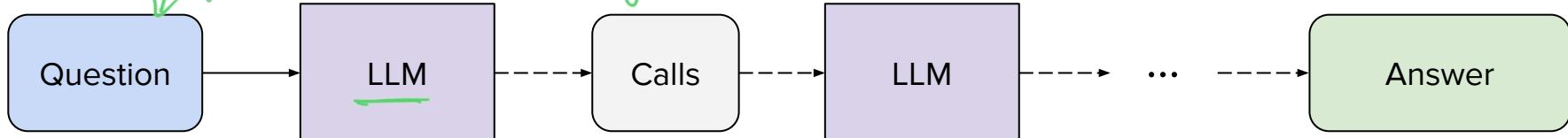
## Traditional



## Reasoning

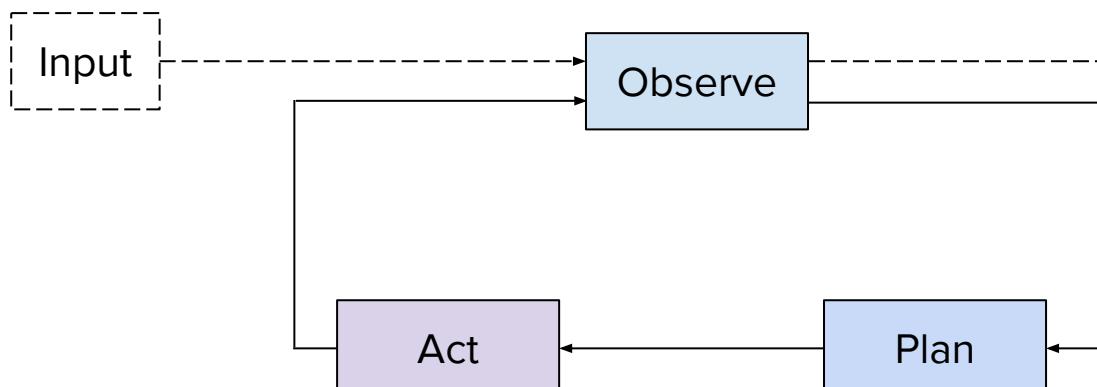


## Agent



# Overview

**ReAct** = Reason + Act



decomposes possible loops  
into different stages  
(we can't get goal by just  
putting query so, ReAct  
helps to decompose  
it  
for optional  
actions  
↓  
decomposes into  
observe + plan + act

# ReAct in action

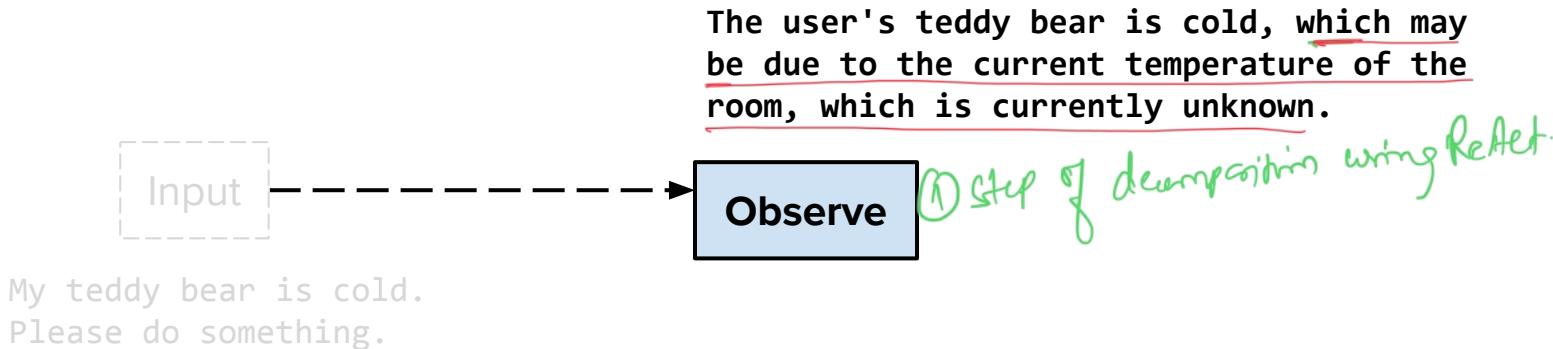


My teddy bear is cold.  
Please do something.

Examples:

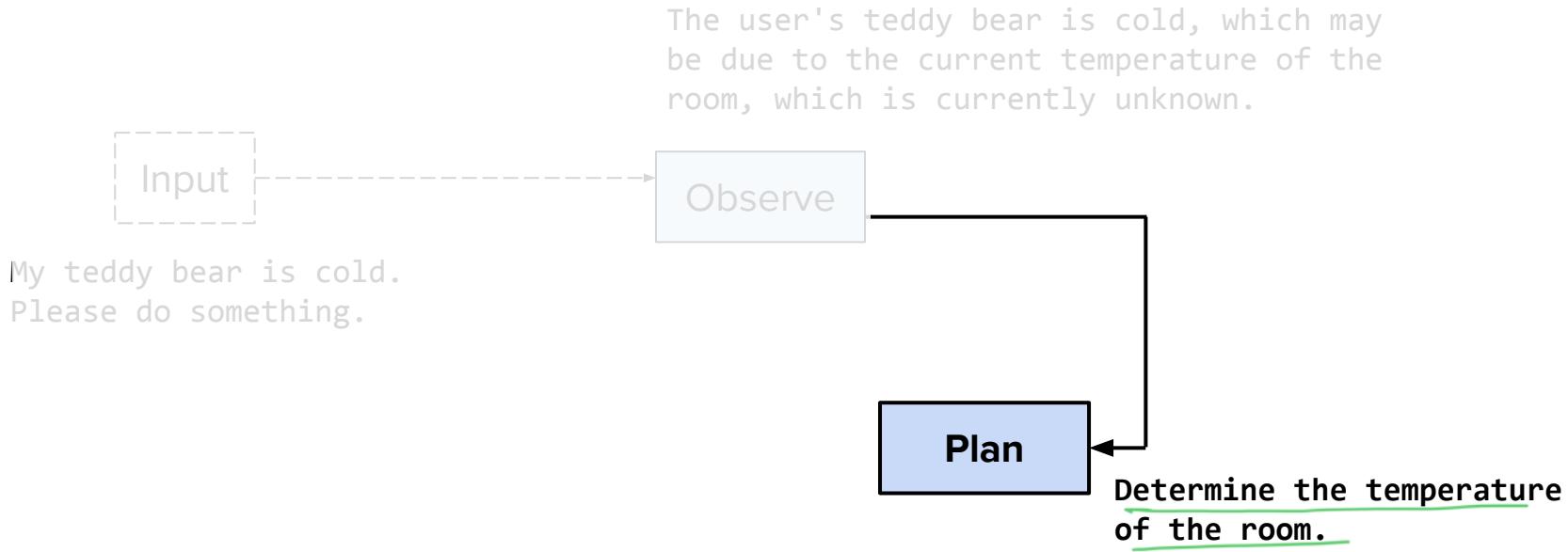
- Manually entered (e.g. user question)
- External event (e.g. metric going beyond a threshold)

# ReAct in action



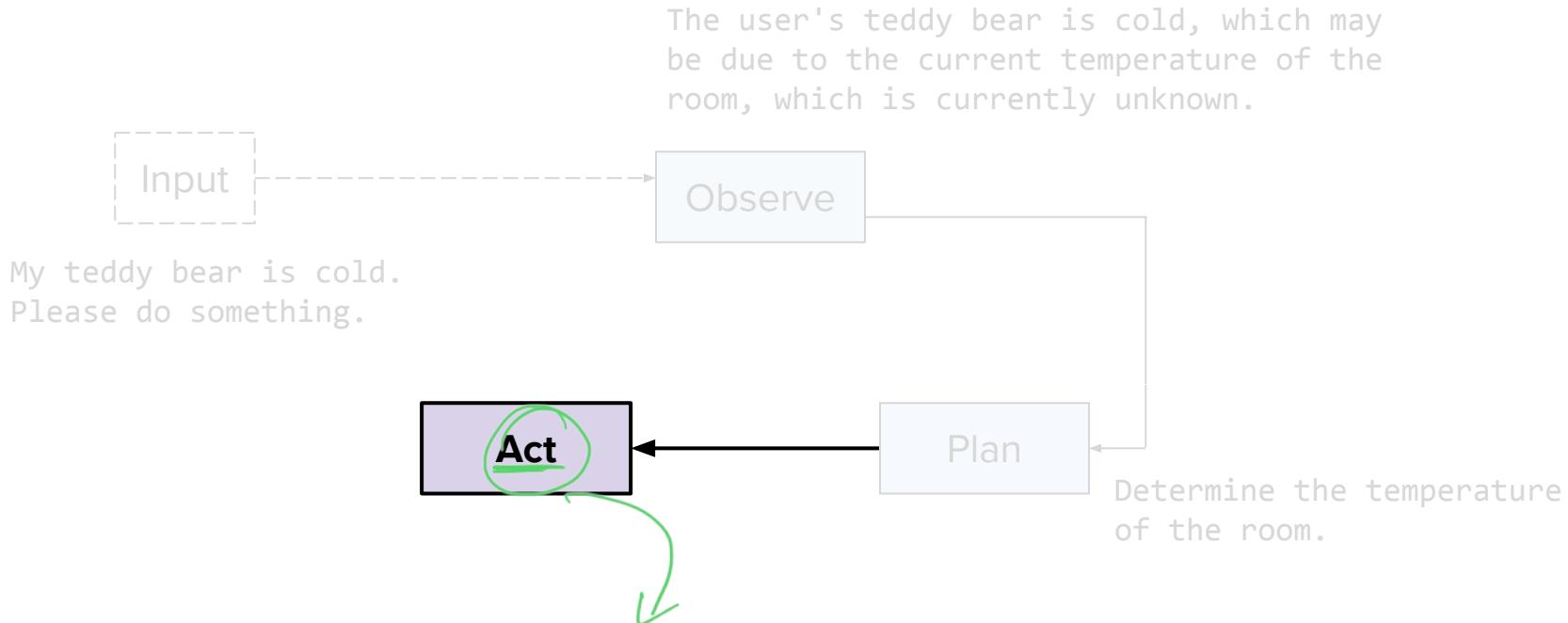
- **Synthesize** previous actions + explicitly **state** what is **currently known** including own knowledge
- **Reasoning**-heavy step to figure out **what is needed**

# ReAct in action



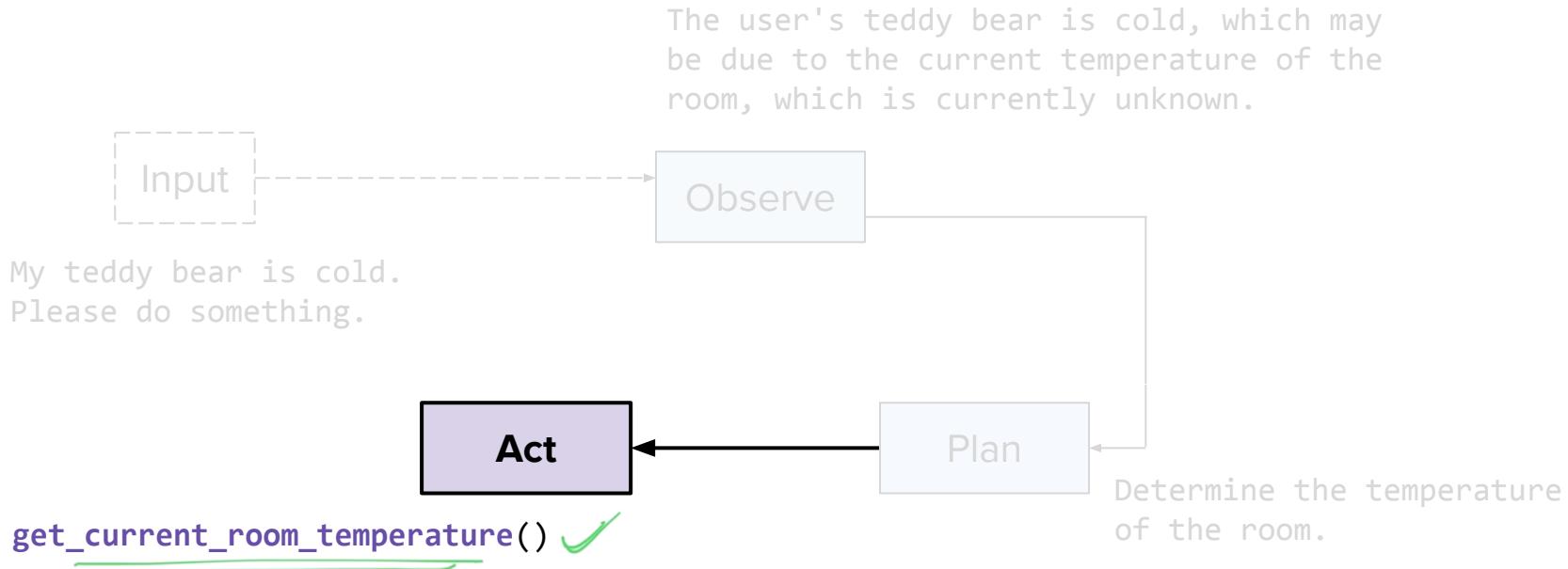
Detail **what tasks** need to be accomplished and **what tools** to call

# ReAct in action



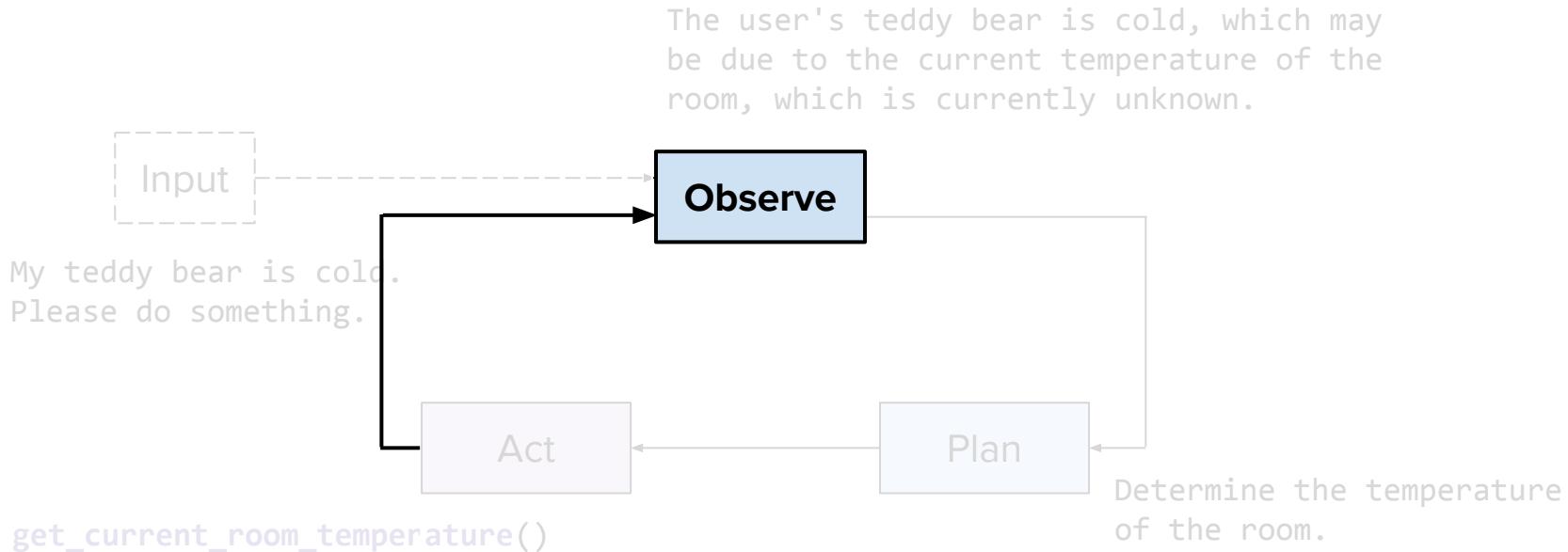
- Perform an action via an **API**
- Look for info in a **database** of documents

# ReAct in action

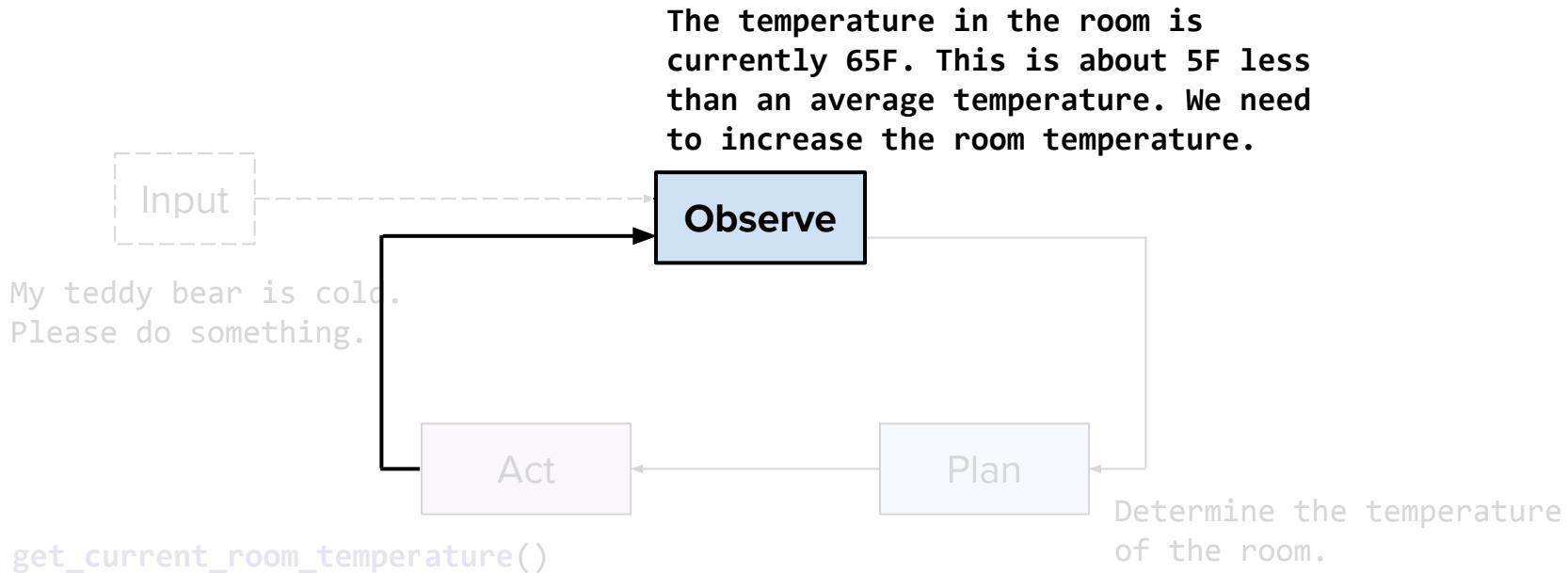


- Perform an action via an **API**
- **Look** for info in a **database** of documents

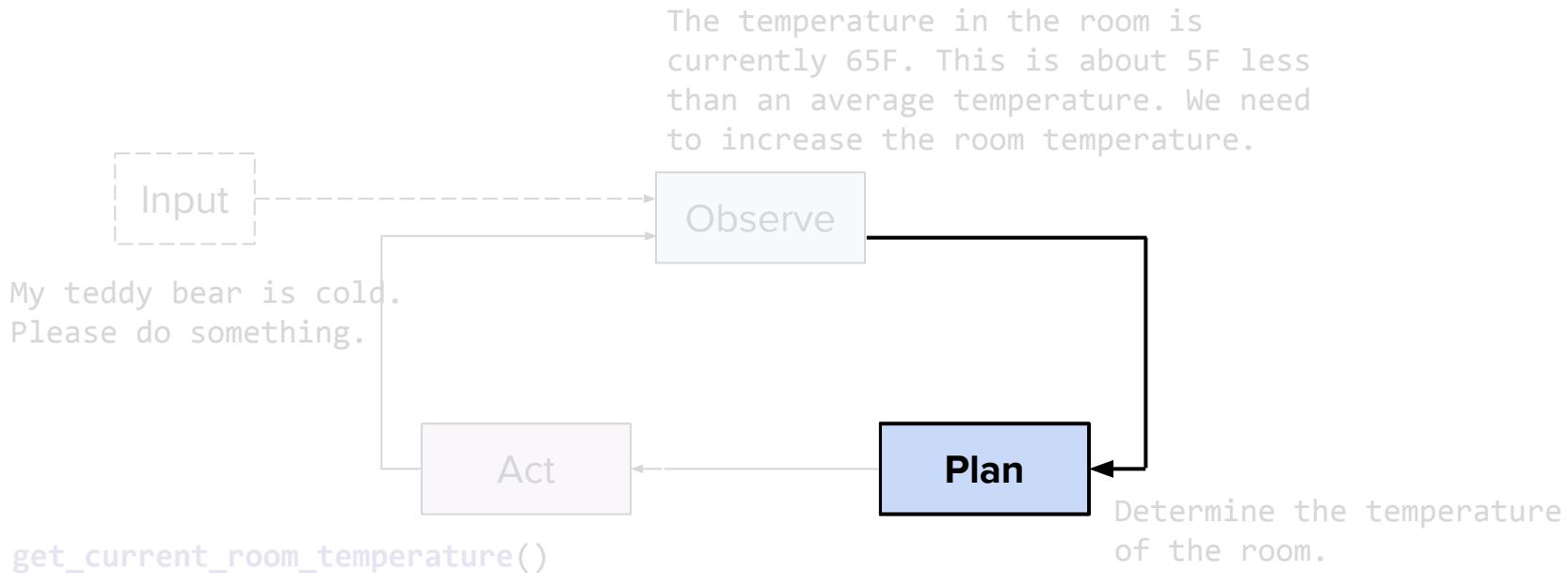
# ReAct in action



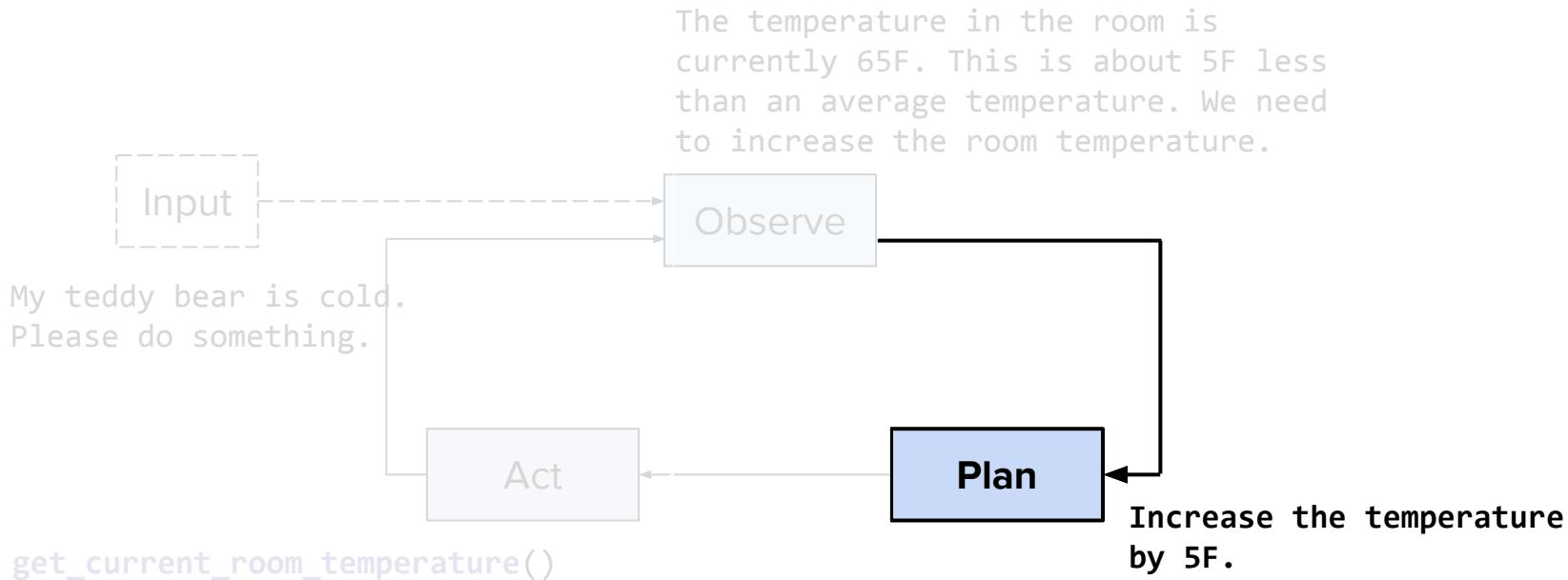
# ReAct in action



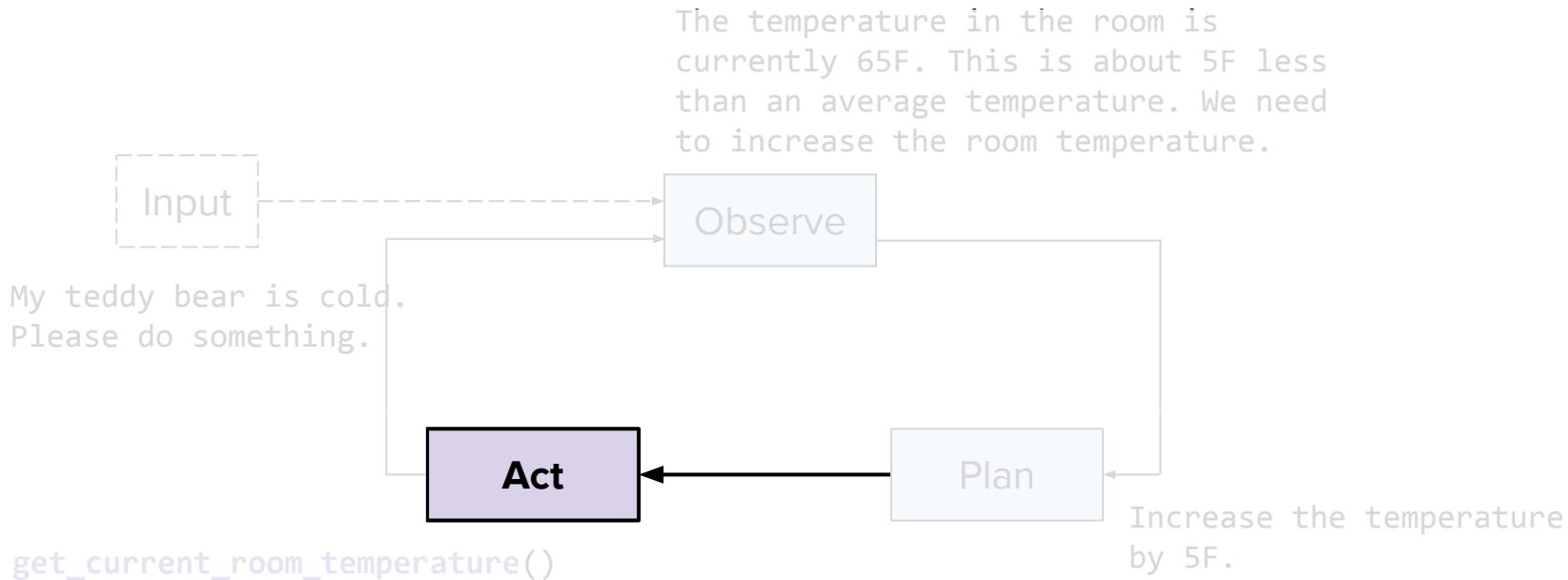
# ReAct in action



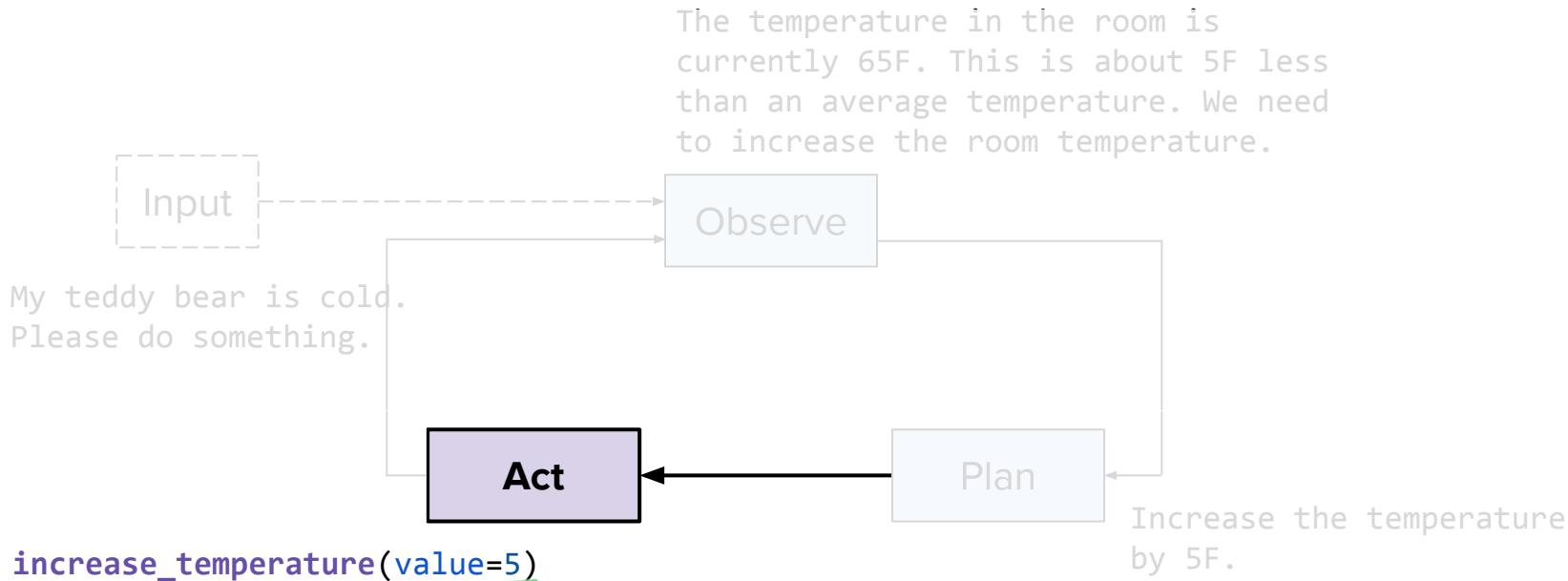
# ReAct in action



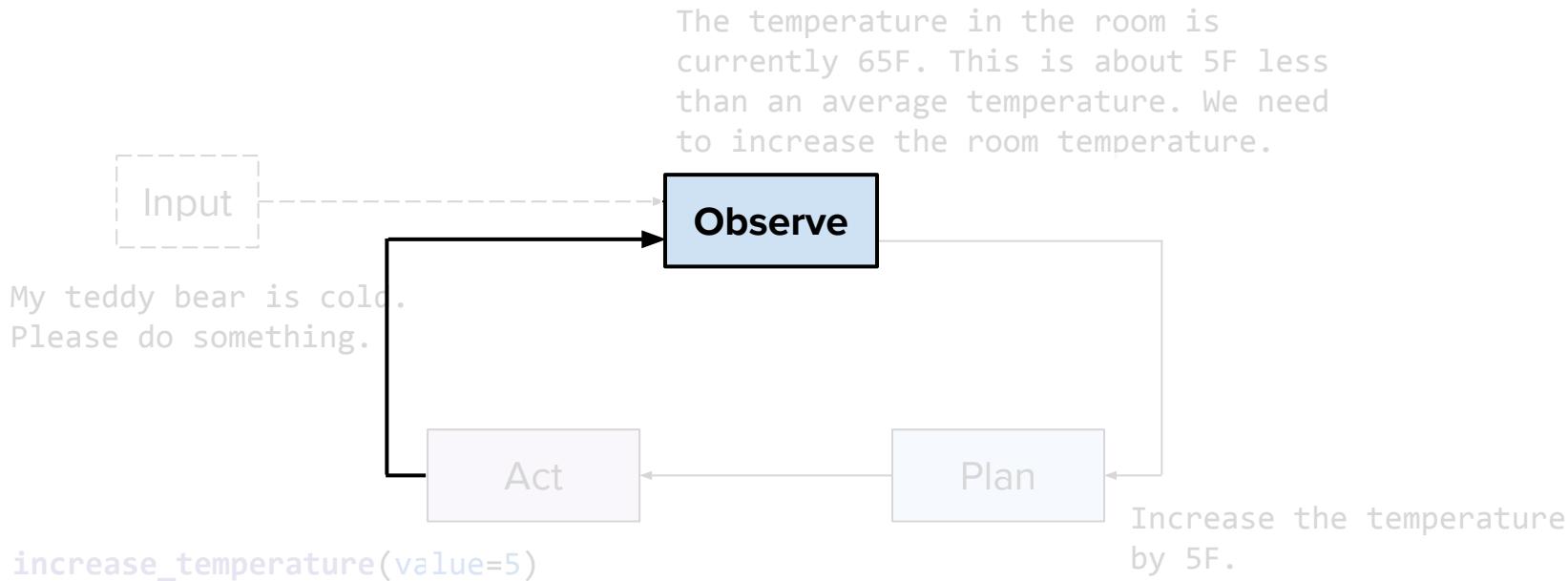
# ReAct in action



# ReAct in action

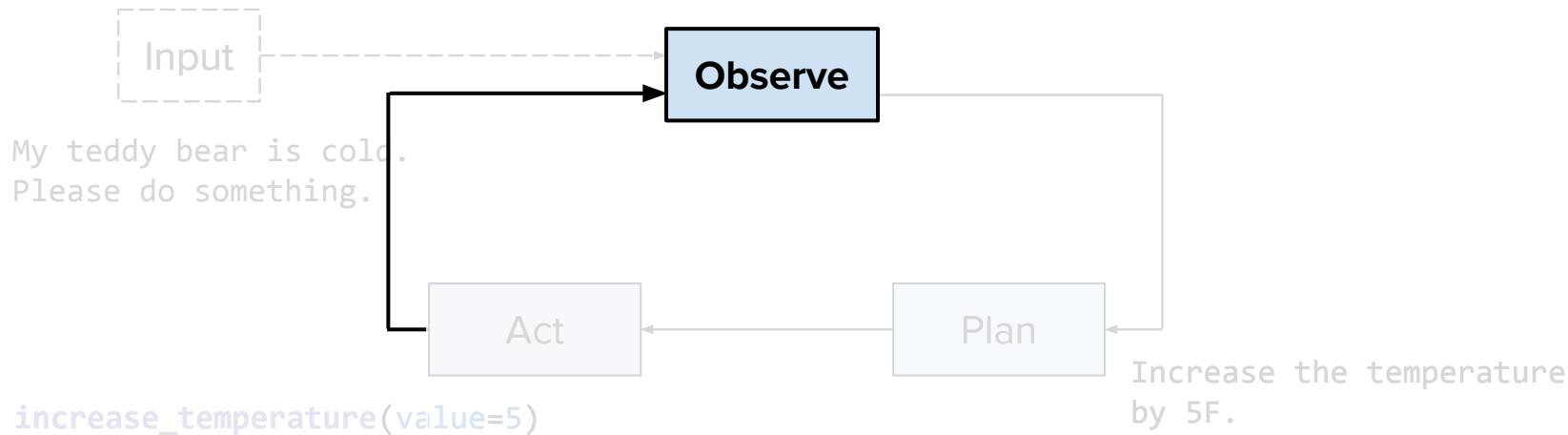


# ReAct in action

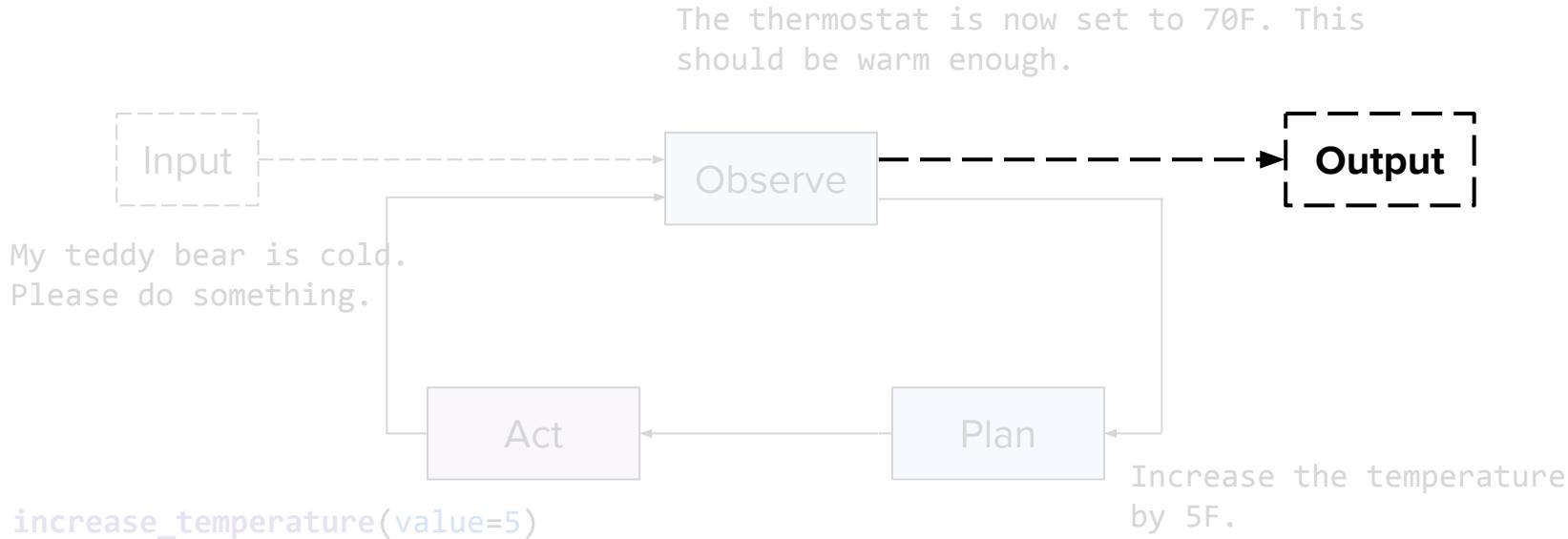


# ReAct in action

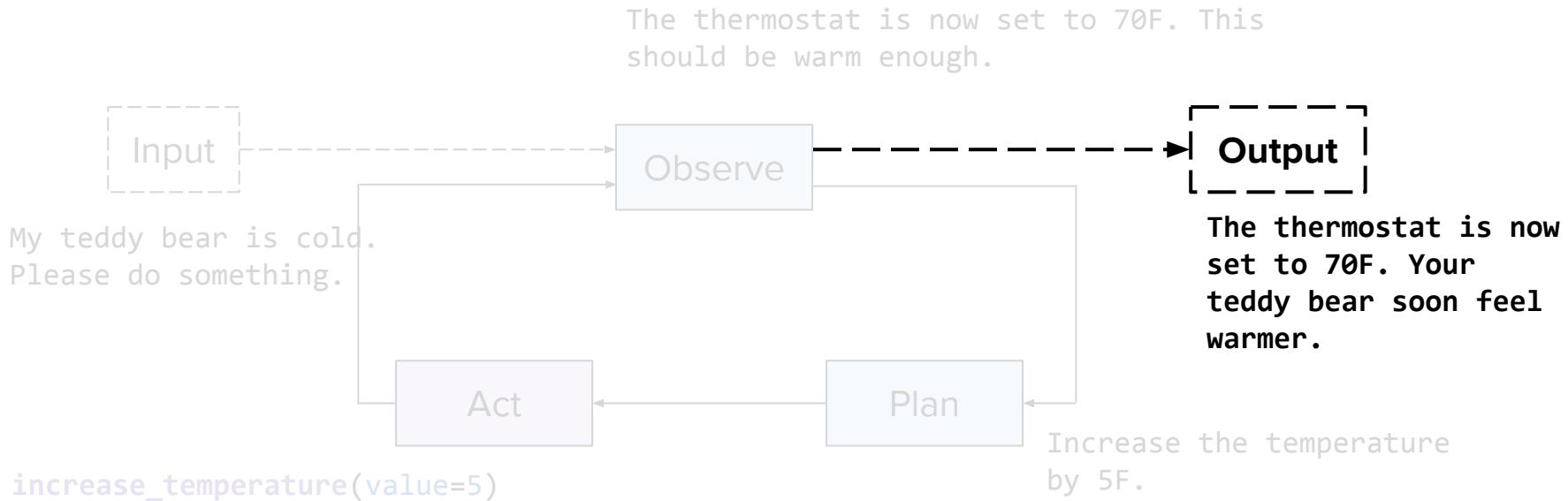
The thermostat is now set to 70F. This should be warm enough.



# ReAct in action



# ReAct in action



# "Agentic" view



# There can be agents for many things

→ multiple agents\*

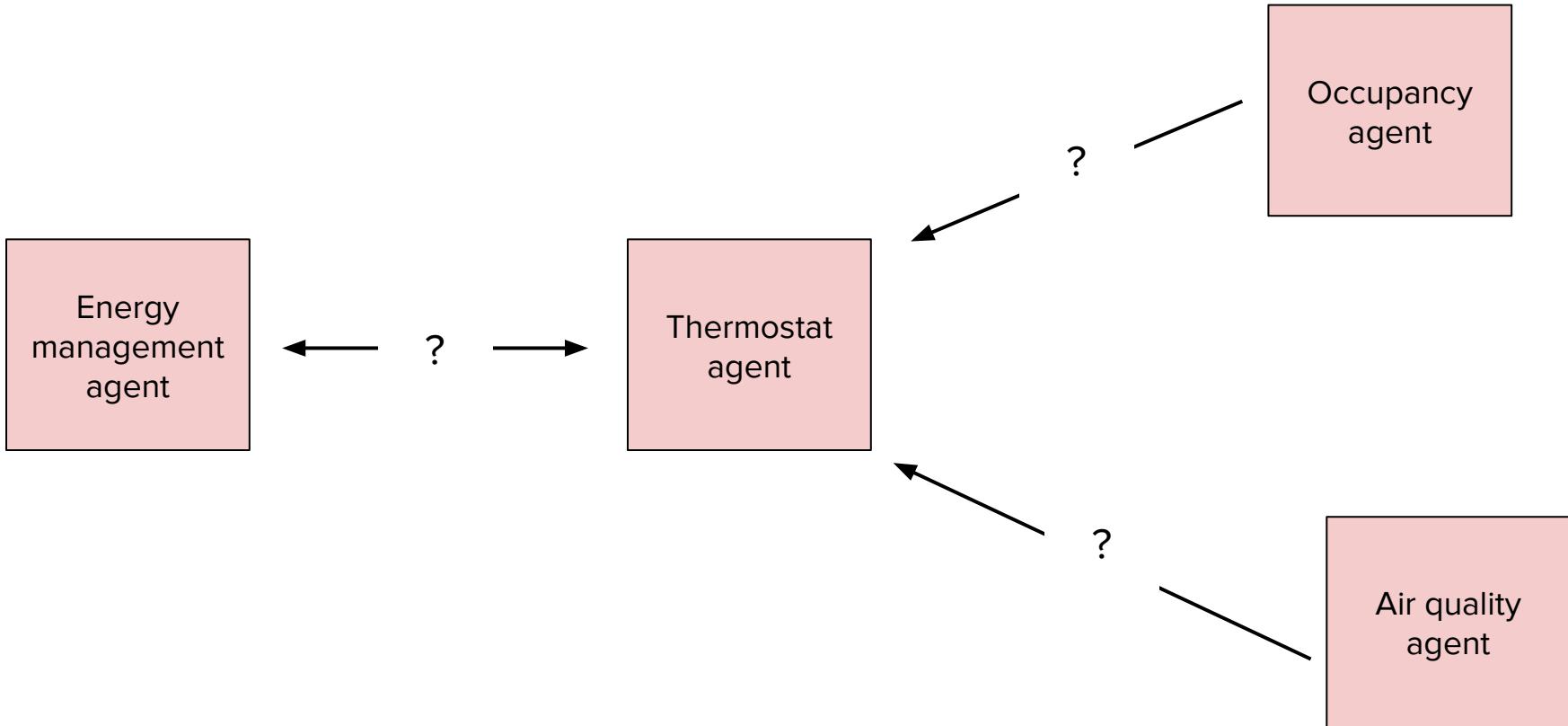
Occupancy  
agent

Energy  
management  
agent

Thermostat  
agent

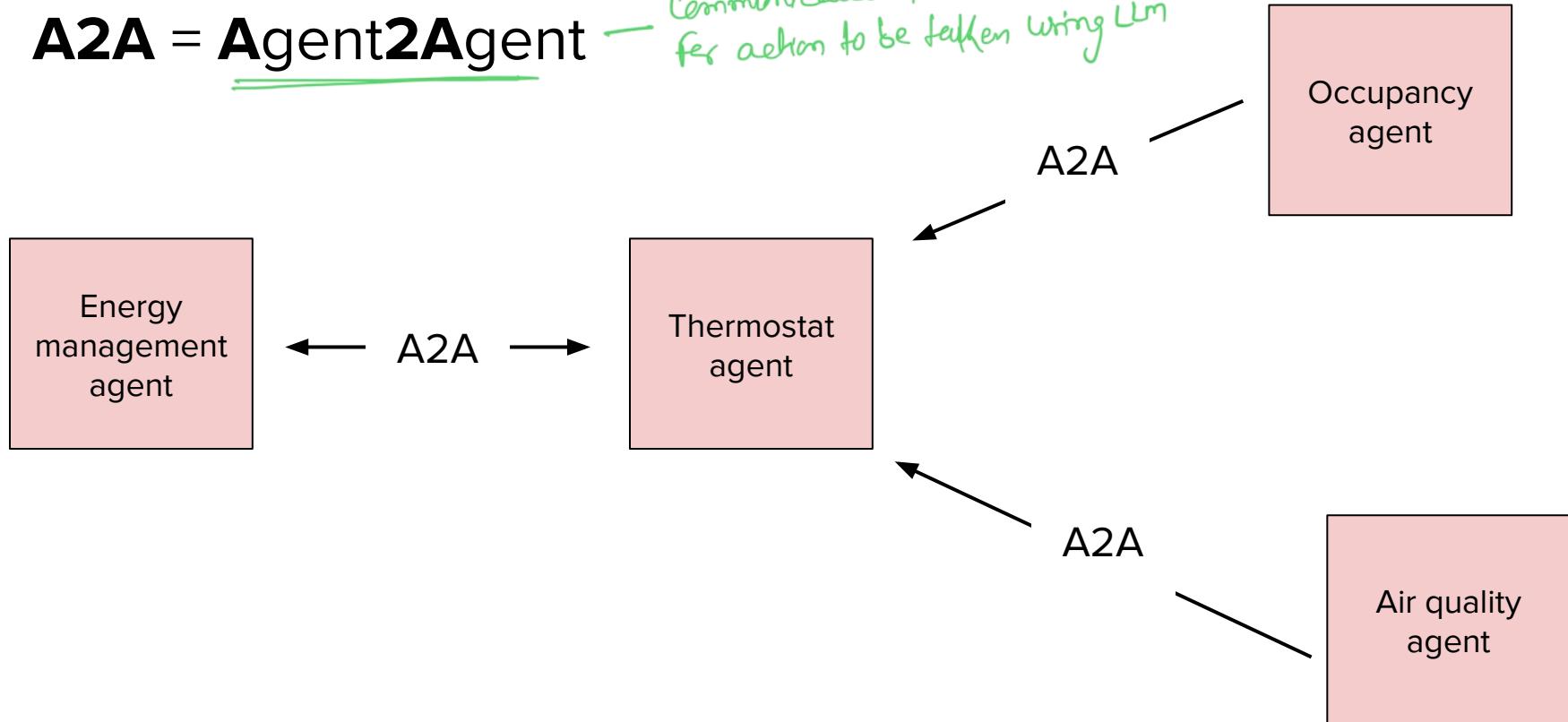
Air quality  
agent

# How can agents communicate with each other?

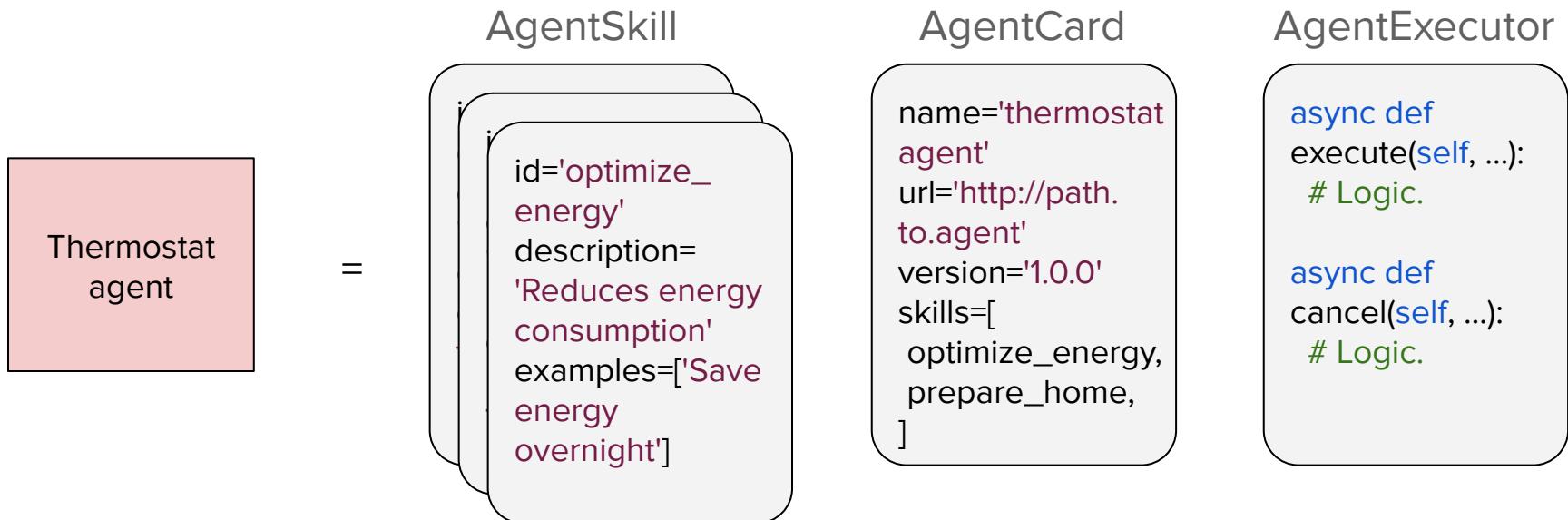


# Standardization: A2A

**A2A = Agent2Agent** — *Communicated b/w multiple agents  
for action to be taken using LLM*



# Standardization: A2A



# Safety

## Risks.

- Potential for harm in the real world
- Example: data exfiltration

# Safety

## Risks.

- Potential for harm in the real world
- Example: data exfiltration

## Remediations.

- Training steps
- Inference safeguards
- Benchmarks, e.g. [Agent-SafetyBench](#)

# Safety

## Risks.

- Potential for harm in the real world
- Example: data exfiltration

## Remediations.

- Training steps
- Inference safeguards
- Benchmarks, e.g. [Agent-SafetyBench](#)

**...very important topic!**

# Just yesterday in the news

Policy

## Disrupting the first reported AI-orchestrated cyber espionage campaign

Nov 13, 2025 • 7 min read

[Read the report](#)

A white tote bag with a keyhole cutout on a red background. The bag has black handles. The red background is a large rectangular area behind the bag.

# Closing thoughts

- **Hallucination** is a (big) problem

A handwritten note in green ink, with a green bracket above it, pointing to the word "Hallucination". The note reads "risk of hallucinated thought process."

# Closing thoughts

- **Hallucination** is a (big) problem
- **Reasoning** abilities are a bottleneck
  - Finetuning helps, but hard
  - New capabilities are very welcome

# Closing thoughts

- **Hallucination** is a (big) problem
- **Reasoning** abilities are a bottleneck
  - Finetuning helps, but hard
  - New capabilities are very welcome
- **Evaluation** is challenging

# Closing thoughts

- **Hallucination** is a (big) problem
- **Reasoning** abilities are a bottleneck
  - Finetuning helps, but hard
  - New capabilities are very welcome
- **Evaluation** is challenging
- Good to **start simple, then iterate and progressively scale up**
- Good to **start with capable models, optimize on size later**

# Closing thoughts

- **Hallucination** is a (big) problem
- **Reasoning** abilities are a bottleneck
  - Finetuning helps, but hard
  - New capabilities are very welcome
- **Evaluation** is challenging
- Good to **start simple**, then **iterate and progressively scale up**
- Good to **start with capable models, optimize on size later**
- **Transparency / observability** helps with user trust and debuggability

# Bonus: AI agents in your daily life

Personal favorite use case: coding!

The screenshot shows a code editor interface with a Python file named `find_teddy_bear.py` open. The file contains code for finding the nearest teddy bear based on GPS coordinates. A red box highlights a floating window titled "Check correctness of find\_teddy\_bear.py". The window displays a message from the AI agent asking for review and lists several issues found in the code. A red arrow points from the word "agent" in the top right corner to the floating window.

```
from dataclasses import dataclass
from geopy.distance import geodesic
import requests

@dataclass
class TeddyBearInfo:
 name: str
 distance_meters: float
 mood: str
 message: str

def find_teddy_bear(location: tuple[float, float]) -> TeddyBearInfo:
 """
 Finds the nearest teddy bear to the given GPS coordinates.

 Parameters:
 location: A (latitude, longitude) pair representing the user's current location.

 Returns:
 TeddyBearInfo: Information about the nearest teddy bear found.
 """
 # Call API to get the closest teddy bear
 user_lat, user_lon = location
 api_url = "https://api.to.teddy.bears.com/v1/closest"
 try:
 response = requests.get(
 api_url,
 params={"latitude": user_lat, "longitude": user_lon},
 timeout=5
)

```

Check correctness of find\_teddy\_bear.py

Can you please check `find_teddy_bear.py` for correctness?

Reviewing `find_teddy_bear.py` for correctness. Checking related files for context.

Explored 5 files

**Issues found**

1. Missing API response validation
  - Assumes the response has `coords`, `name`, and `mood`. Missing keys will raise `KeyError`.
  - No check that `coords` is a 2-element sequence.
2. Type hint compatibility
  - `tuple[float, float]` requires Python 3.9+. For older versions, use `Tuple[float, float]` from `typing`.
3. Incomplete error handling
  - Only catches `RequestException`, `KeyError`, `ValueError`, `Typ`

Plan, @ for context, / for commands

Auto

ICME

Thank you for your attention!

---