Lecture 7

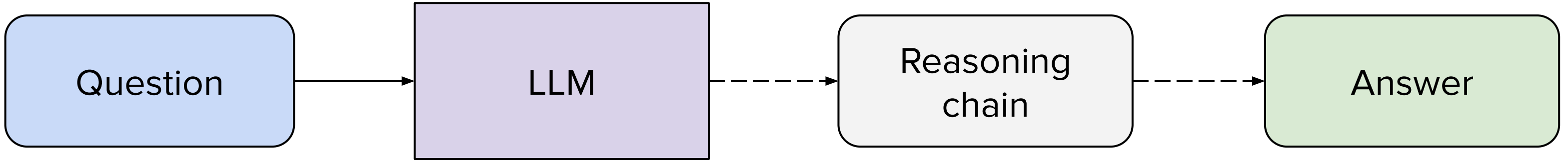
CME 295: Transformers & Large Language Models

 **Afshine Amidi & Shervine Amidi** 

Stanford University

Recap of last episode...

Reasoning models



Recap of last episode...

Reasoning models

GRPO

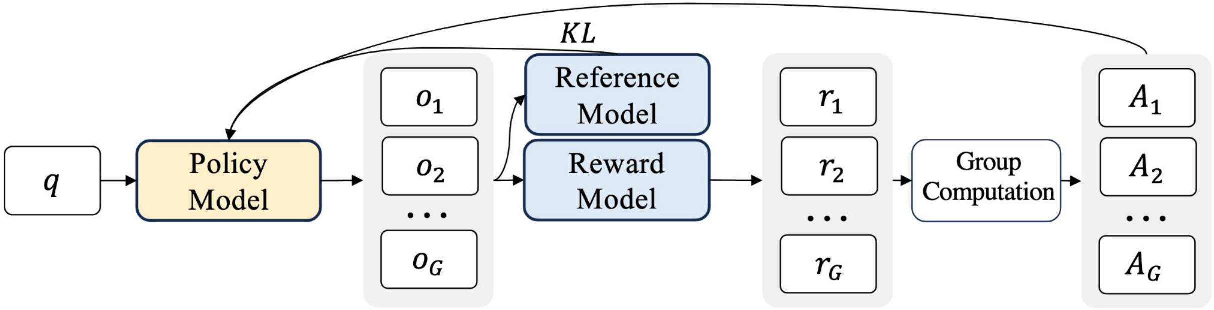
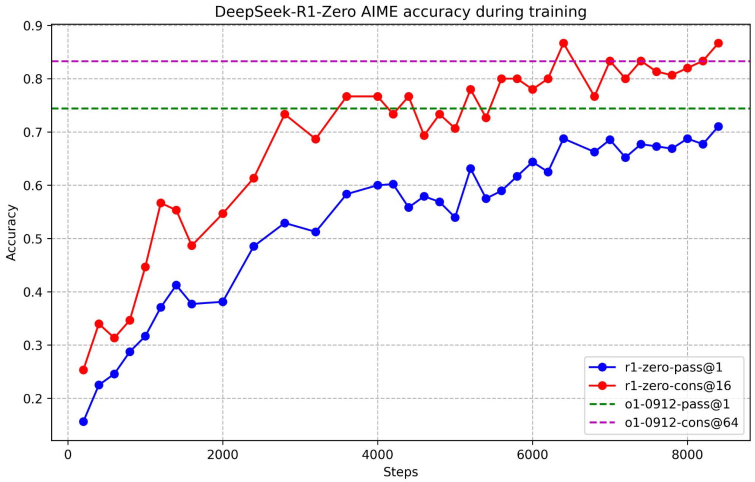
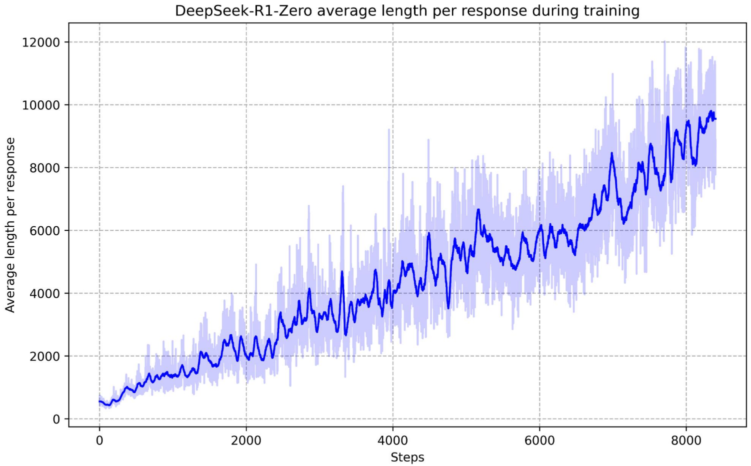


Figure from "DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models", Shao et al., 2024.

Recap of last episode...





Recap of last episode...

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

Stanford University

"*DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*", *DeepSeek-AI, 2025.*

ICME

Recap of last episode...

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

DAPO $\frac{1}{\sum_{i=1}^{G}|o_i|}\sum_{i=1}^{G}\sum_{t=1}^{|o_i|}$

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 ← Focus of last lecture
- 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 today

Transformers & Large Language Models

RAG

Tool calling

Agents

Stanford University

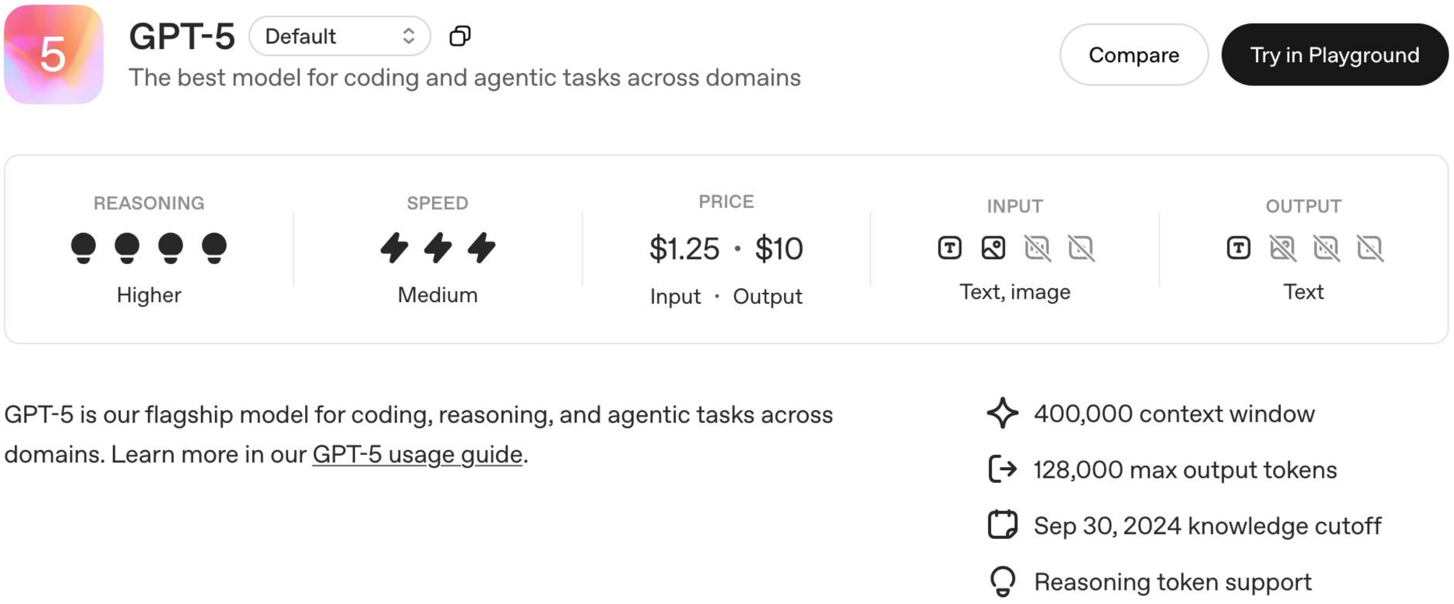
ICME

Motivation

- **Knowledge** of LLM **constrained** to pretraining data

Motivation

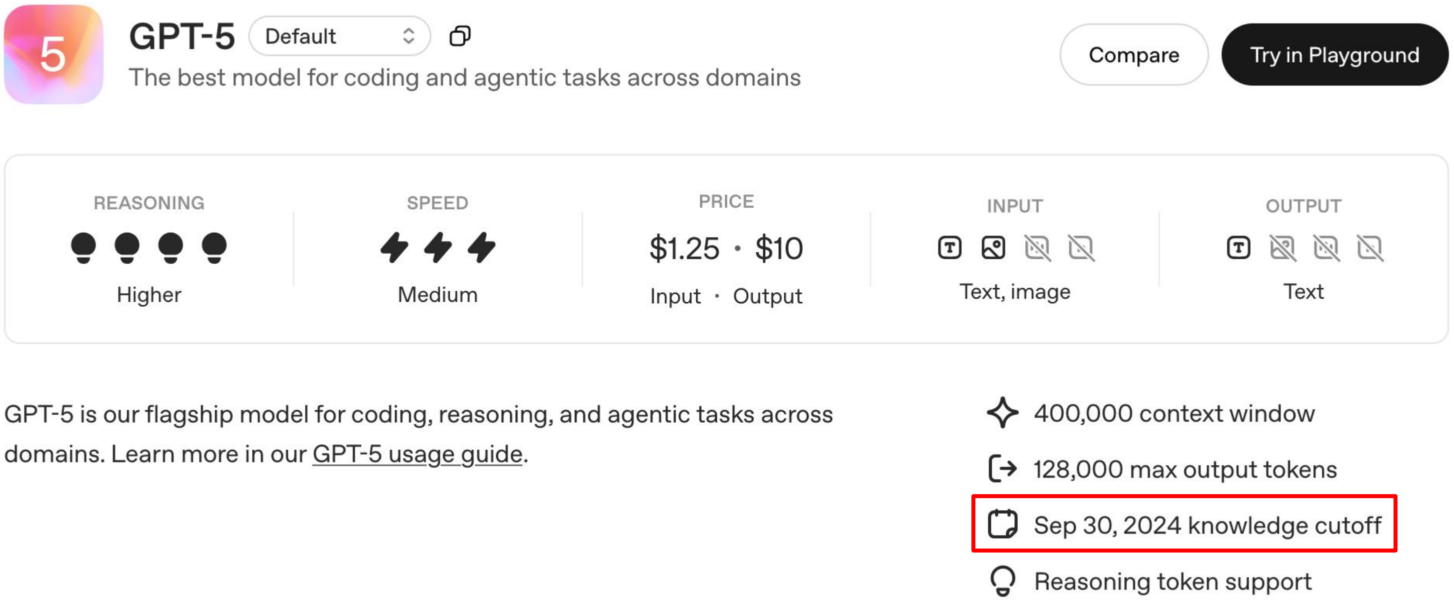
- **Knowledge** of LLM **constrained** to pretraining data



Screenshot of OpenAI's GPT-5 model card, taken on November 9th, 2025.

Motivation

- **Knowledge** of LLM **constrained** to pretraining data



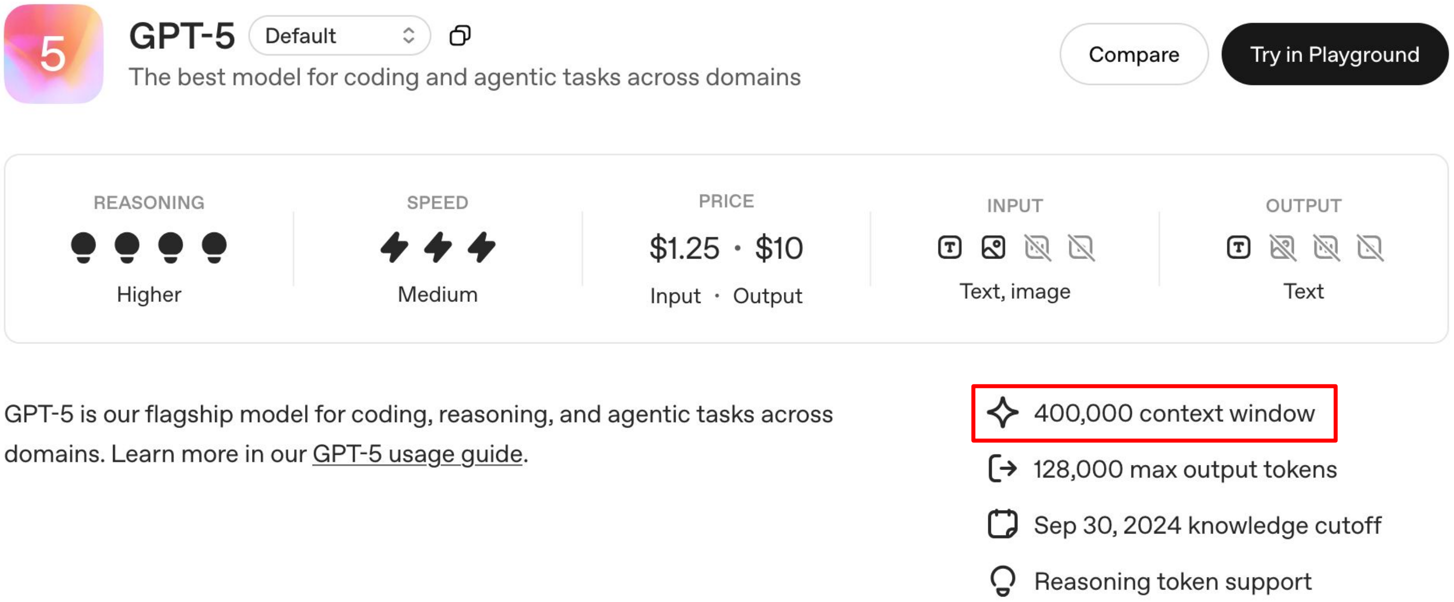
Screenshot of OpenAI's GPT-5 model card, taken on November 9th, 2025.

Motivation

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

Motivation

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



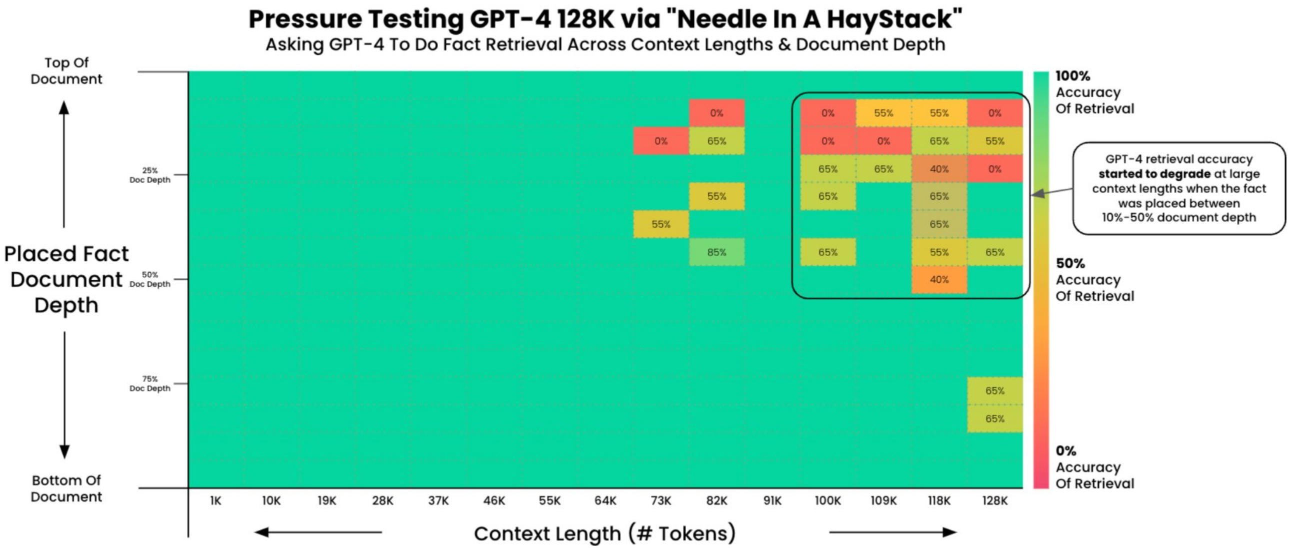
Screenshot of OpenAI's GPT-5 model card, taken on November 9th, 2025.

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



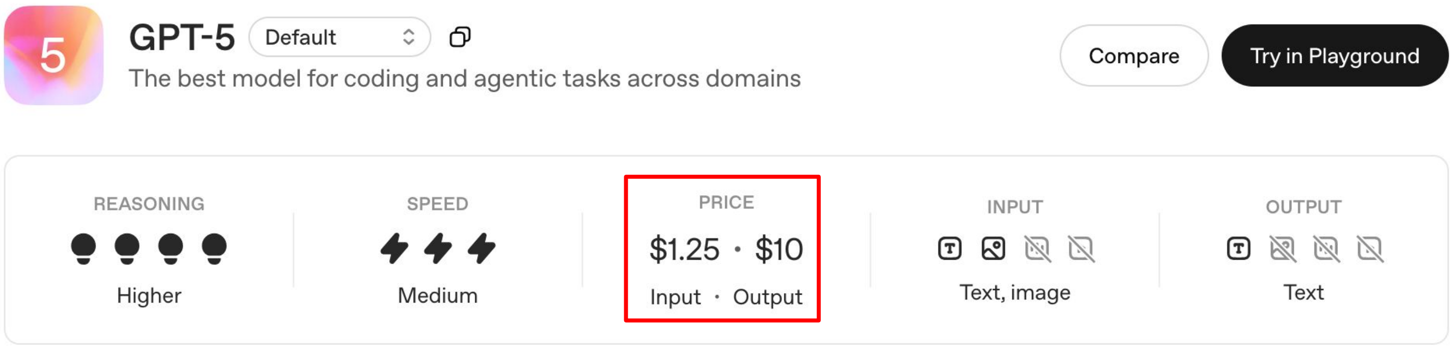
GitHub repo containing "Needle in a Haystack - Pressure Testing LLMs" tests, Kamradt, 2023.

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 = **R**etrieval-**A**ugmented **G**eneration

"Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", Lewis et al., 2020.

Overview

RAG = **R**etrieval-**A**ugmented **G**eneration

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

"Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", Lewis et al., 2020.

Overview

RAG = Retrieval-Augmented Generation

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

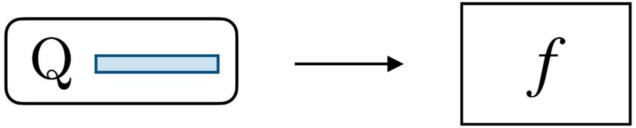


Figure from "Super Study Guide: Transformers & Large Language Models", Amidi et al., 2024.

Overview

RAG = **R**etrieval-**A**ugmented **G**eneration

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

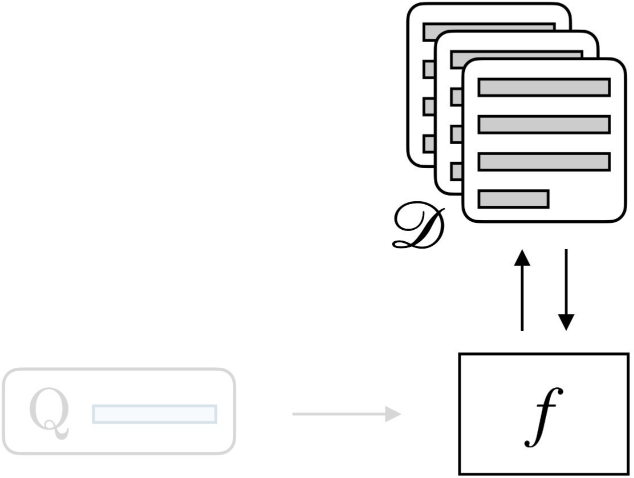


Figure from "Super Study Guide: Transformers & Large Language Models", Amidi et al., 2024.

Overview

RAG = **R**etrieval-**A**ugmented **G**eneration

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

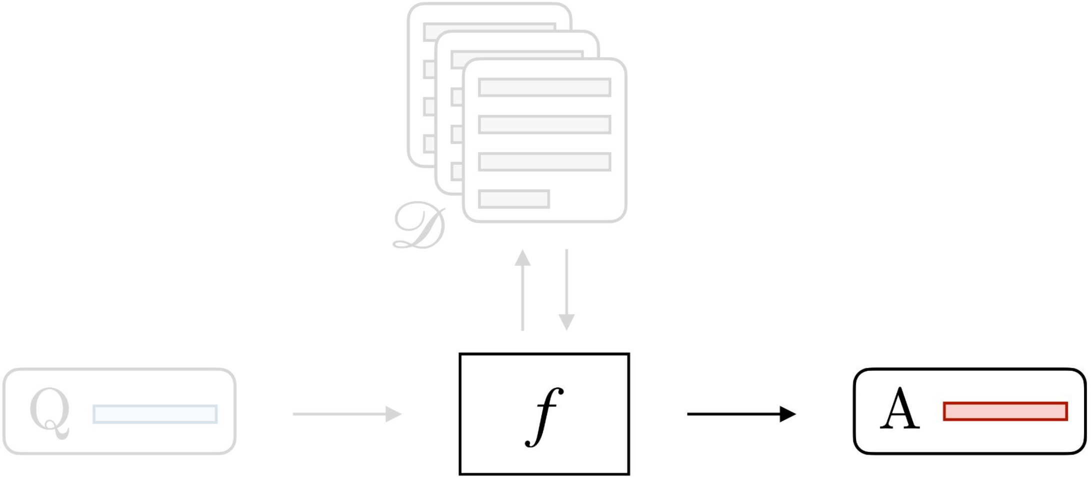
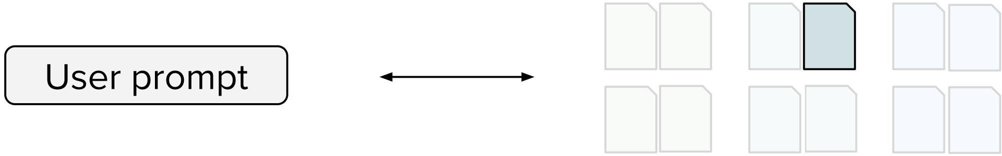


Figure from "Super Study Guide: Transformers & Large Language Models", Amidi et al., 2024.

Retrieve, Generate, Augment

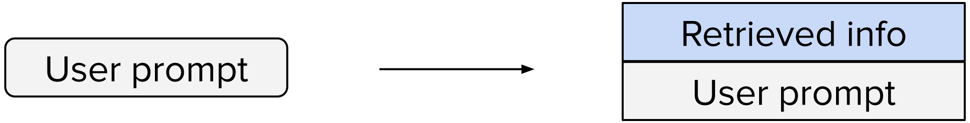
1. **Retrieve** relevant document via similarity operation across the knowledge base



Retrieve, Generate, Augment

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

2. **Augment** prompt with retrieved information

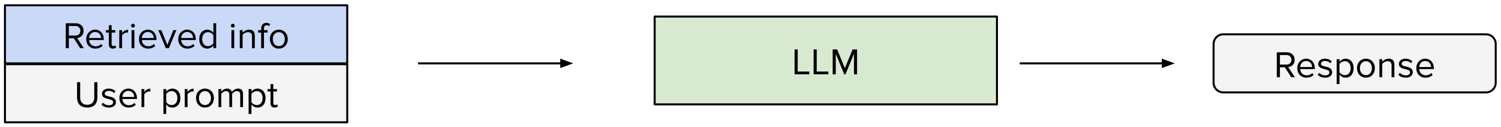


Retrieve, Generate, Augment

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

2. **Augment** prompt with retrieved information

3. **Generate** response



Retrieve, Generate, Augment

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

User prompt ⟷

2. **Augment** prompt with retrieved information

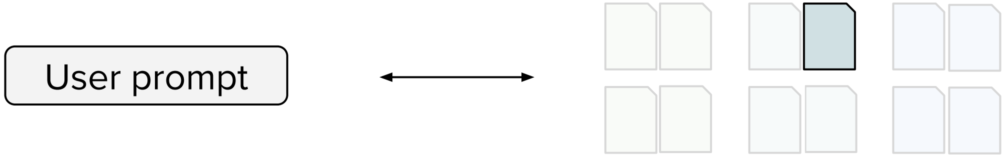
User prompt → Retrieved info / User prompt

3. **Generate** response

Retrieved info / User prompt → LLM → Response

Focus on retrieval stage

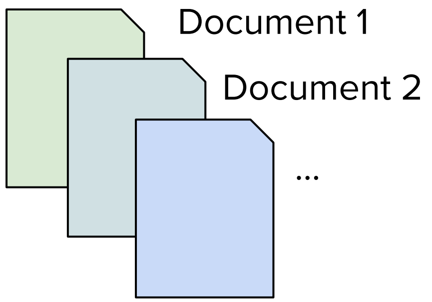
1. **R**etrieve relevant document via similarity operation across the knowledge base



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

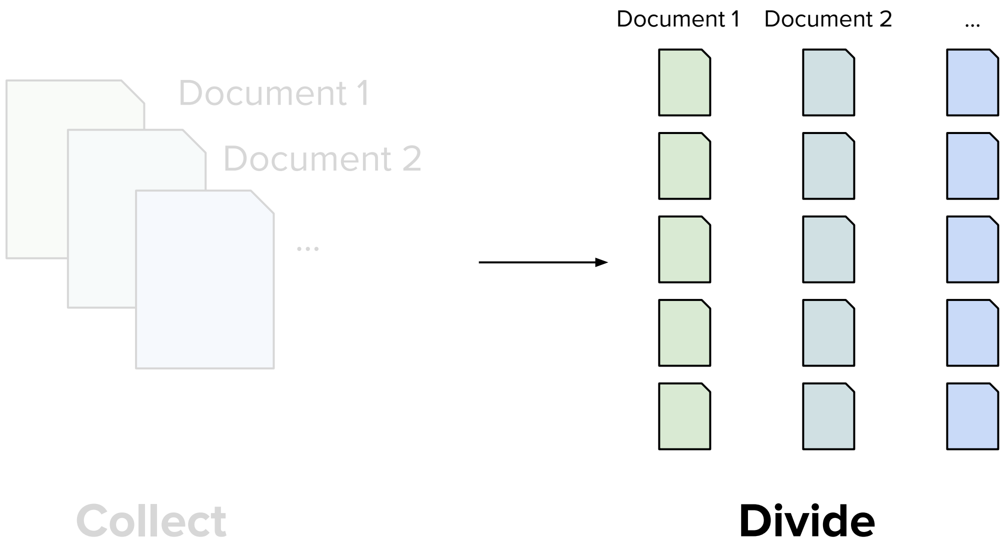
3. **G**enerate response

Prerequisite: Create knowledge base

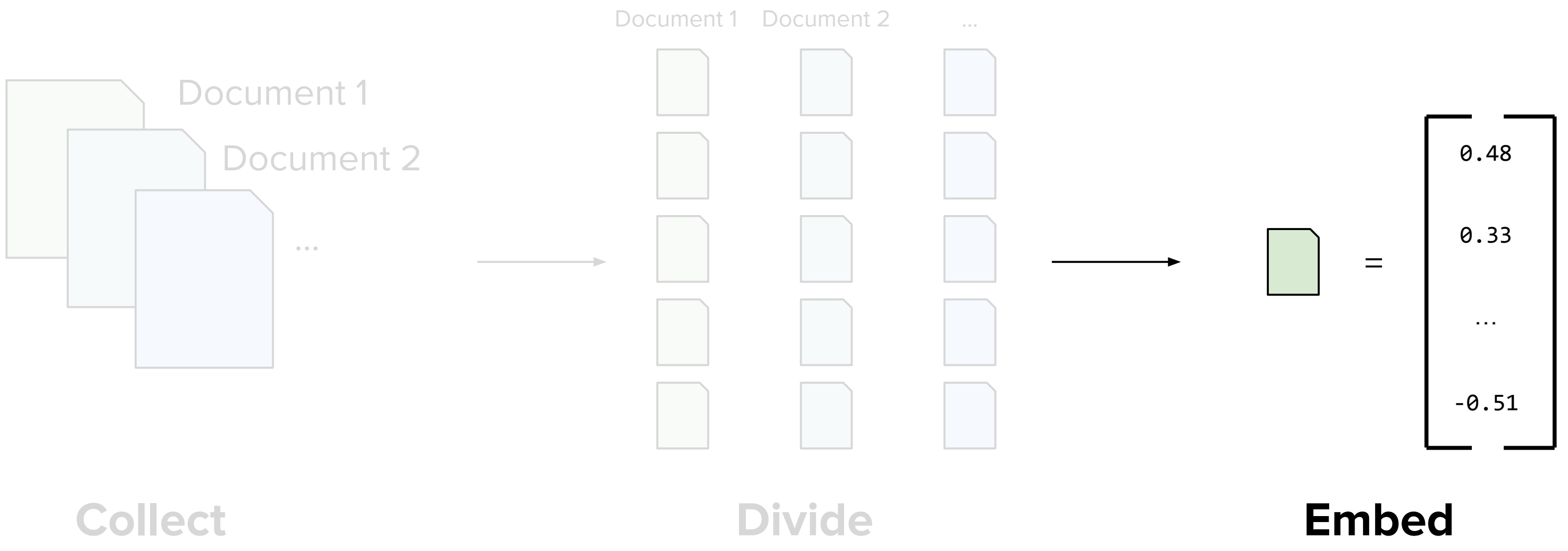


Collect

Prerequisite: Create knowledge base



Prerequisite: Create knowledge base



Prerequisite: Create knowledge base

Hyperparameters. Embedding size, chunk size, overlap between chunks

Retrieval overview

Step 1 – Candidate retrieval: Select potentially-relevant candidates

- Maximize 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

💯 **Step 2 – Ranking**: Give final relevance score

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

Step 1: candidate retrieval

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**

Query

Chunk

Step 1: candidate retrieval

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

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

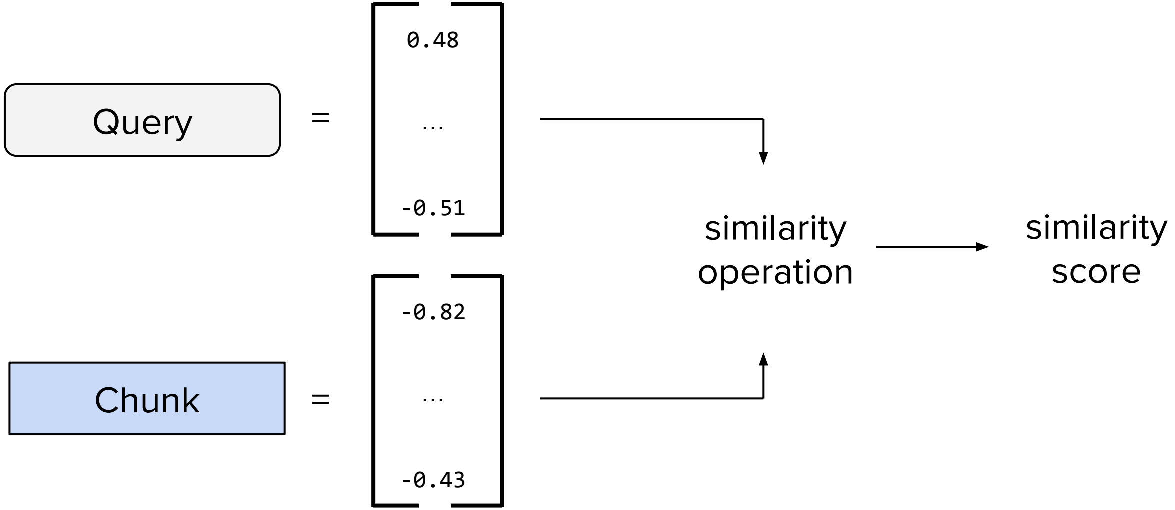
$$\text{Query} = \begin{bmatrix} \texttt{0.48} \\ \cdots \\ \texttt{-0.51} \end{bmatrix}$$

$$\text{Chunk} = \begin{bmatrix} \texttt{-0.82} \\ \cdots \\ \texttt{-0.43} \end{bmatrix}$$

Step 1: candidate retrieval

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

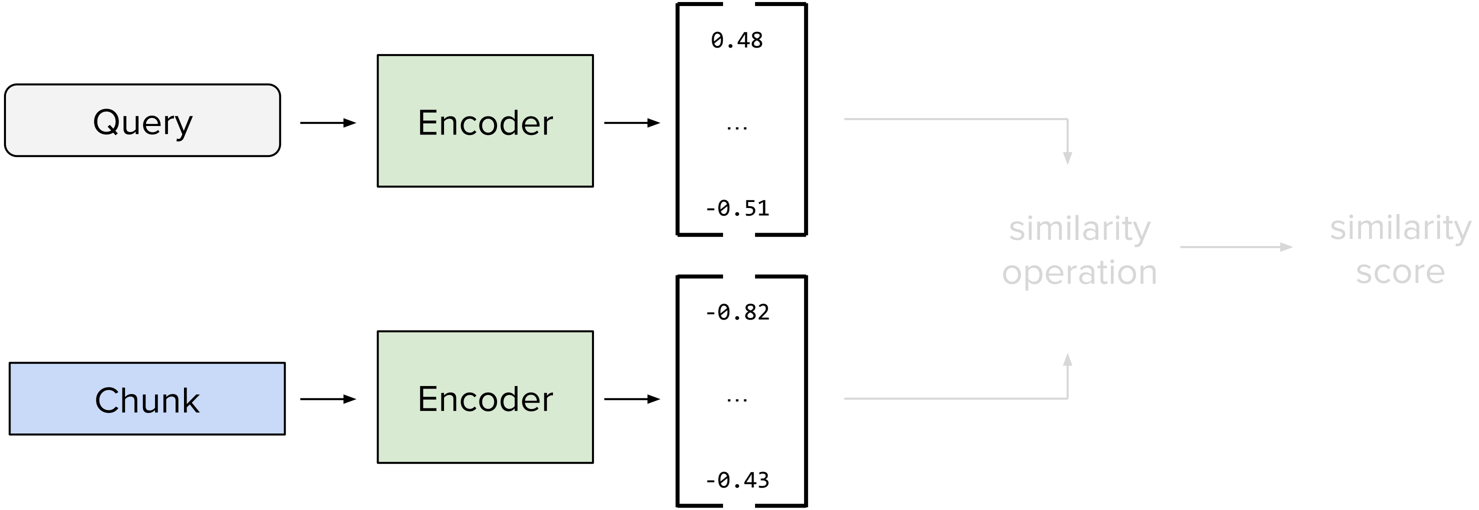
Method 1. Semantic search using **embeddings-based similarity**



Step 1: candidate retrieval

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

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



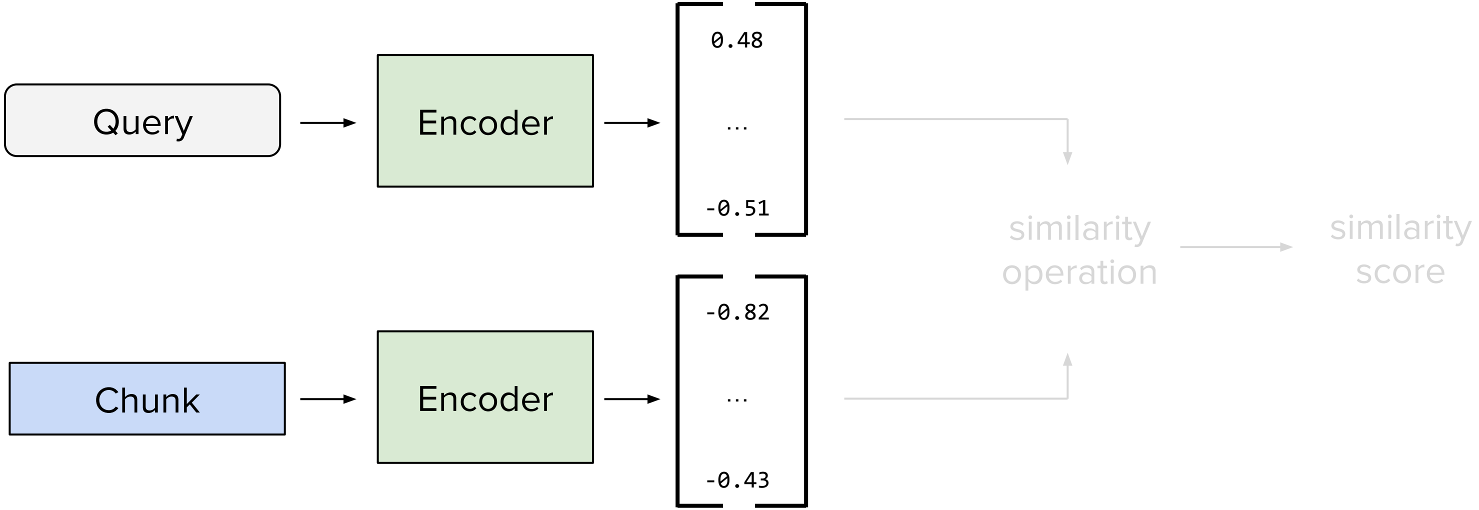
"*Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*", *Reimers et al., 2019.*

Step 1: candidate retrieval

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

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

"bi-encoder"



"*Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*", *Reimers et al., 2019.*

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

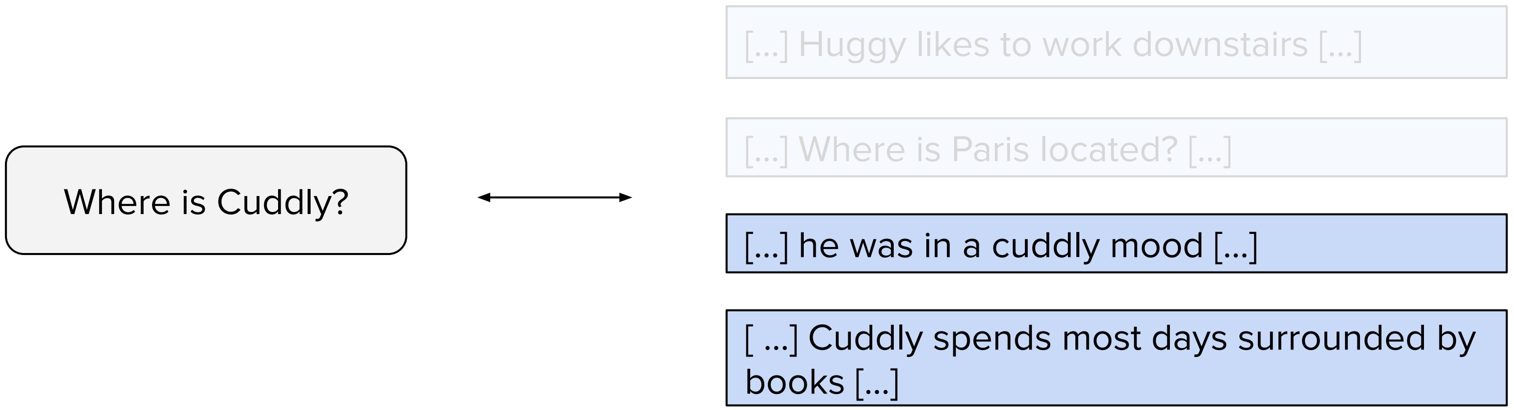
Objective. Select potentially-relevant candidates via search across knowledge base

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

Step 1: candidate retrieval

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

Method 2. Search based on keywords matching via **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

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

"*Precise Zero-Shot Dense Retrieval without Relevance Labels*", *Gao et al., 2022.*

Extensions that can help with initial retrieval

- Mitigate **discrepancy** in nature of embeddings

Prompt

Where is Cuddly?

"Precise Zero-Shot Dense Retrieval without Relevance Labels", Gao et al., 2022.

Extensions that can help with initial retrieval

- Mitigate **discrepancy** in nature of embeddings

Prompt

Where is Cuddly?

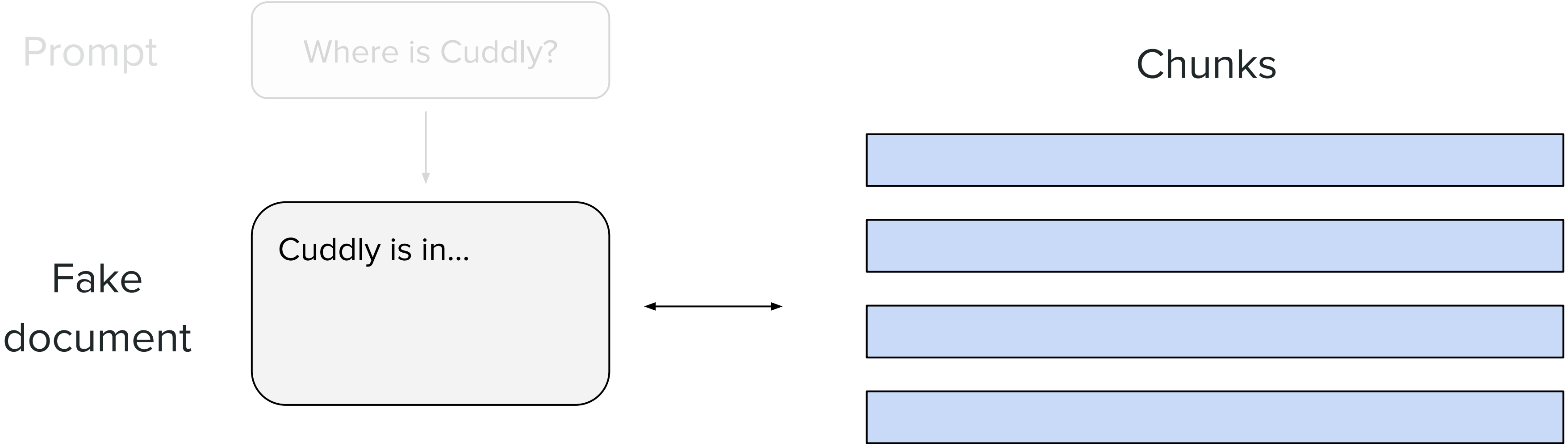
Fake document

Cuddly is in...

"Precise Zero-Shot Dense Retrieval without Relevance Labels", *Gao et al., 2022.*

Extensions that can help with initial retrieval

- Mitigate **discrepancy** in nature of embeddings

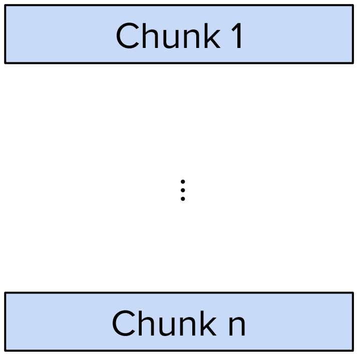


"*Precise Zero-Shot Dense Retrieval without Relevance Labels*", *Gao et al., 2022.*

Extensions that can help with initial retrieval

- Mitigate **discrepancy** in nature of embeddings

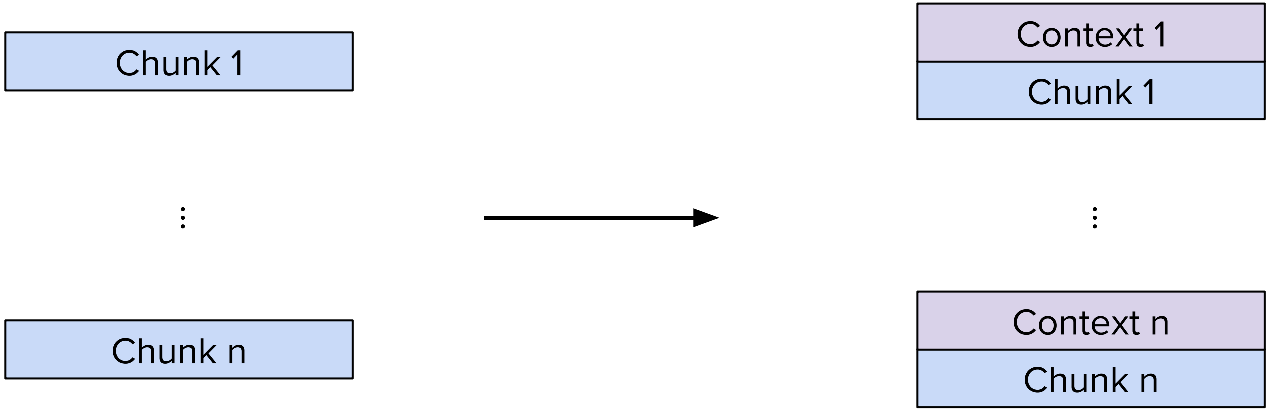
- **Contextualize** document chunks



Introducing Contextual Retrieval, Anthropic team, 2024.

Extensions that can help with initial retrieval

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



Introducing Contextual Retrieval, *Anthropic team, 2024.*

Extensions that can help with initial retrieval

- Mitigate **discrepancy** in nature of embeddings

- **Contextualize** document chunks

```
<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.
```


Introducing Contextual Retrieval, *Anthropic team, 2024.*

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.
```


Introducing Contextual Retrieval, Anthropic team, 2024.

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

OpenAI model pricing page, screenshot taken on November 10th, 2025.

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

OpenAI model pricing page, screenshot taken on November 10th, 2025.

Step 2: ranking

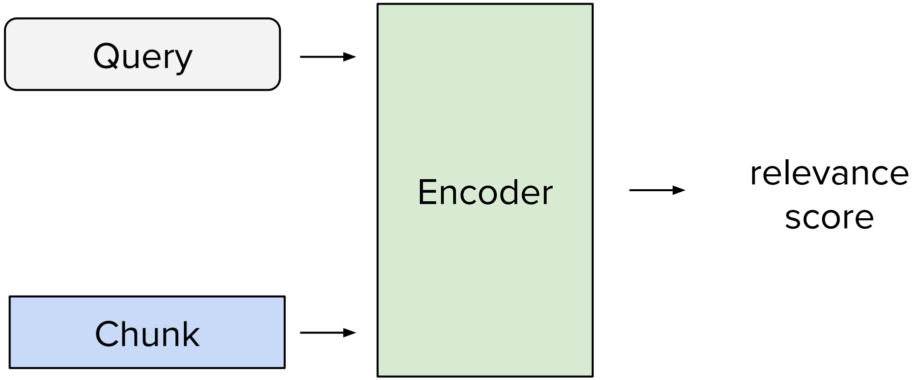
Objective. Provide final relevance score using more sophisticated (re-)ranker

Query

Chunk

Step 2: ranking

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

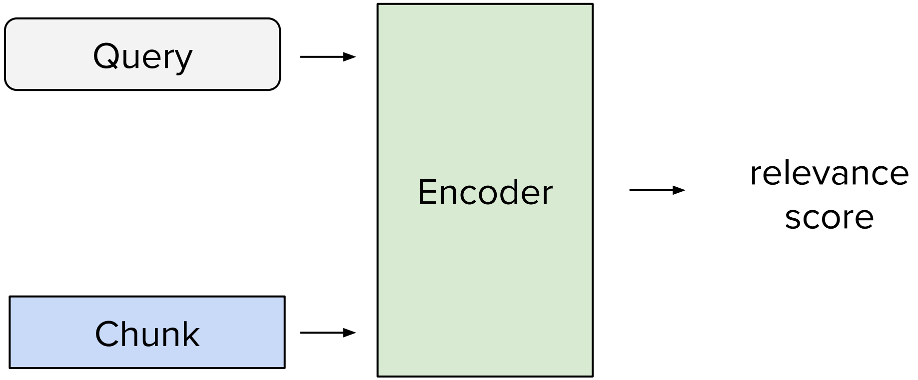


Suggested reading: "Cross-Encoders", SBERT.net

Step 2: ranking

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

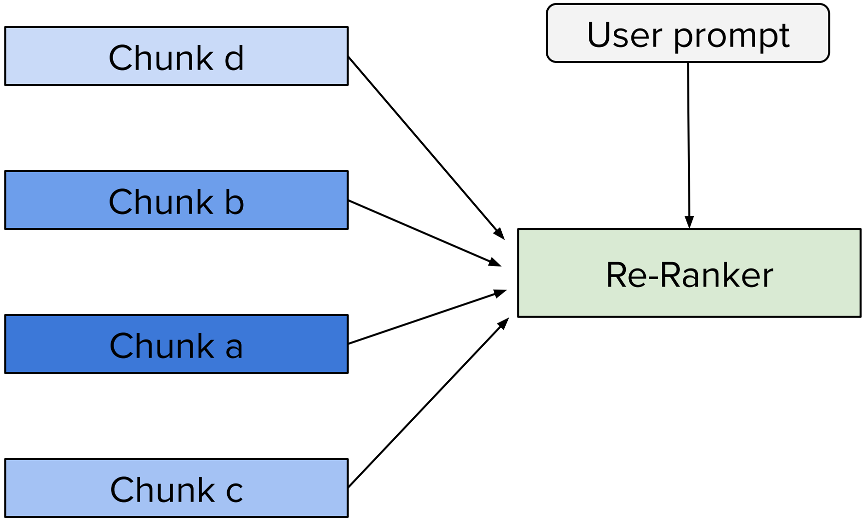
"cross-encoder"



Suggested reading: "Cross-Encoders", SBERT.net

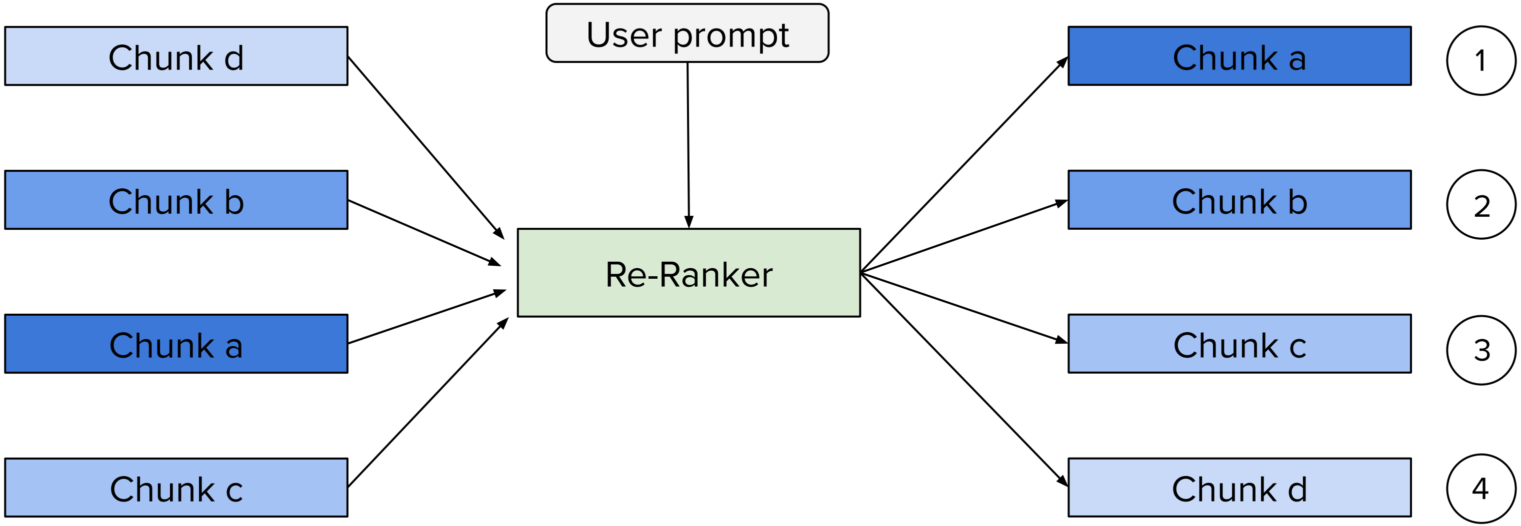
Step 2: ranking

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



Step 2: ranking

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

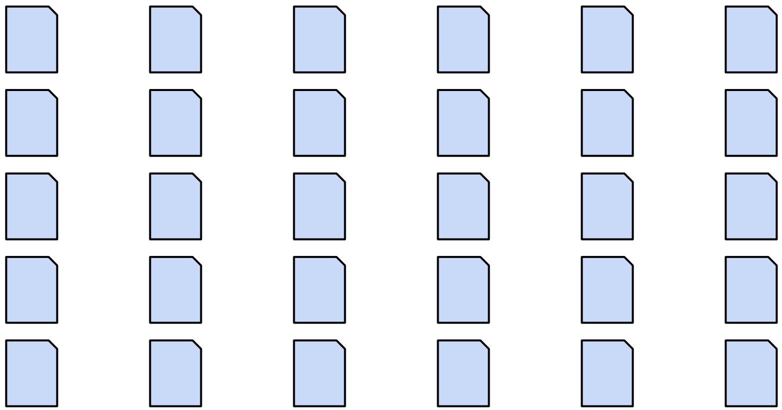


Quantify performance of retrieval

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

Quantify performance of retrieval

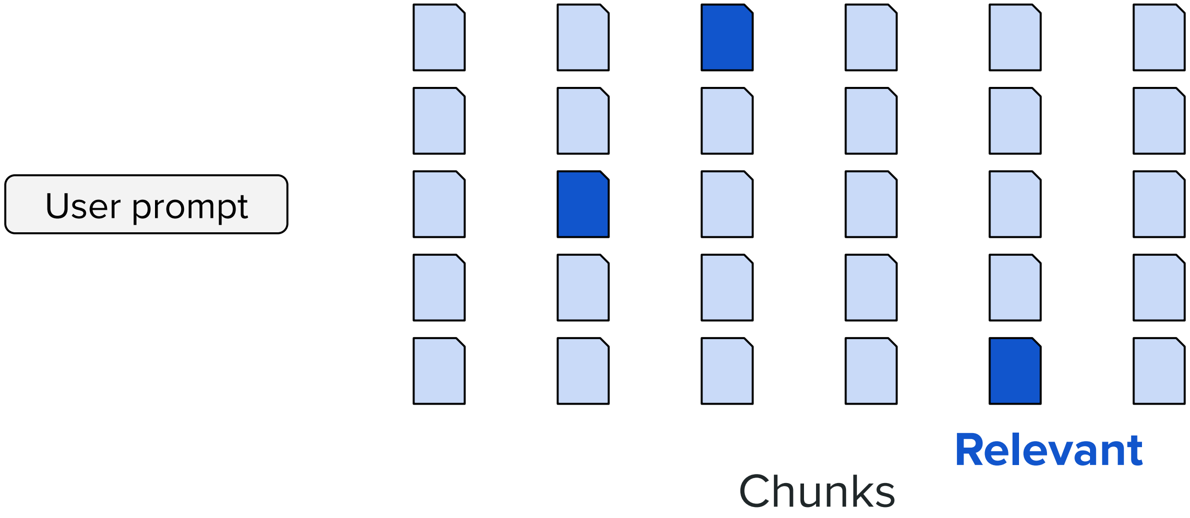
Setup. Evaluate if **retrieved chunks** are **relevant**



Chunks

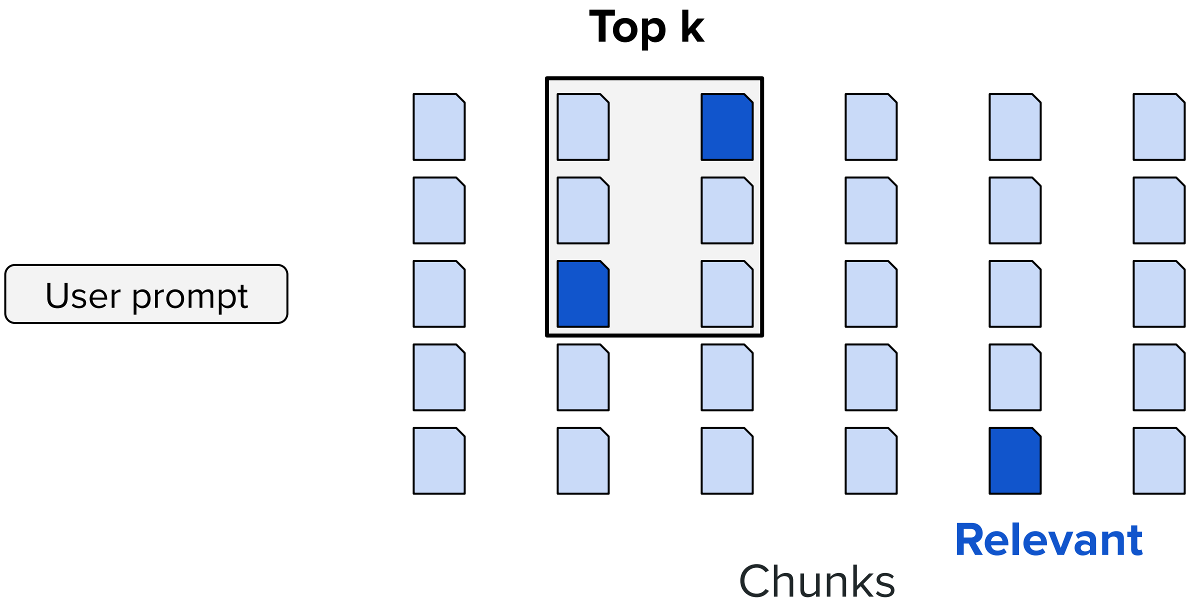
Quantify performance of retrieval

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



Quantify performance of retrieval

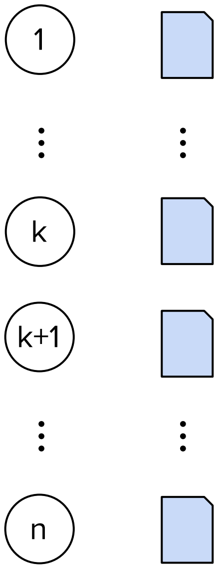
Setup. Evaluate if **retrieved chunks** are **relevant**



Quantify performance of retrieval

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

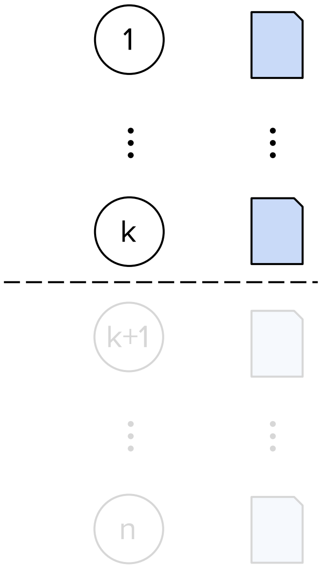
Ranking



Quantify performance of retrieval

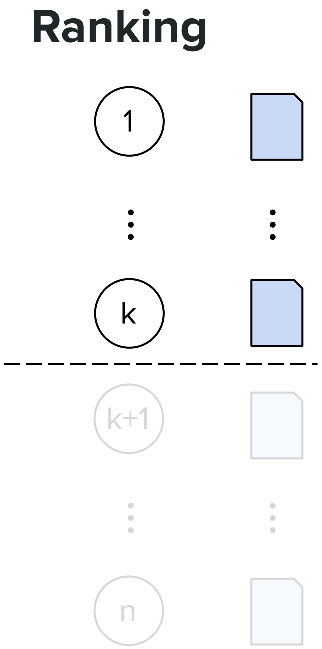
Setup. Evaluate if **retrieved chunks** are **relevant**

Ranking



Quantify performance of retrieval

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

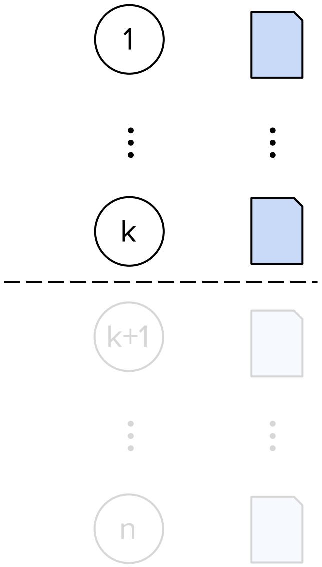


- **N**ormalized **D**iscounted **C**umulative **G**ain at **k** (**NDCG@k**)

Quantify performance of retrieval

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

Ranking



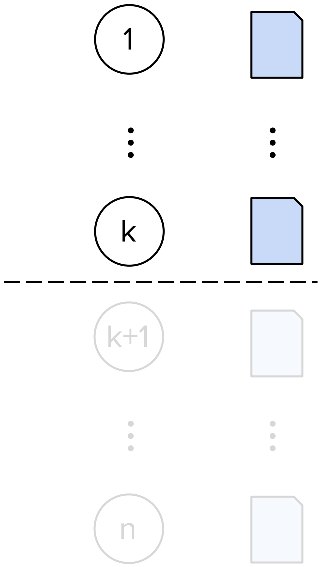
- **N**ormalized **D**iscounted **C**umulative **G**ain at **k** (**NDCG@k**)

$$\boxed{\mathrm{DCG@}k = \sum_{i=1}^{k} \frac{\mathrm{rel}_i}{\log_2(i+1)}} \qquad \text{with} \quad \mathrm{rel}_i \in \{0, 1\}$$

Quantify performance of retrieval

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

Ranking



- **N**ormalized **D**iscounted **C**umulative **G**ain at **k** (**NDCG@k**)

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

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

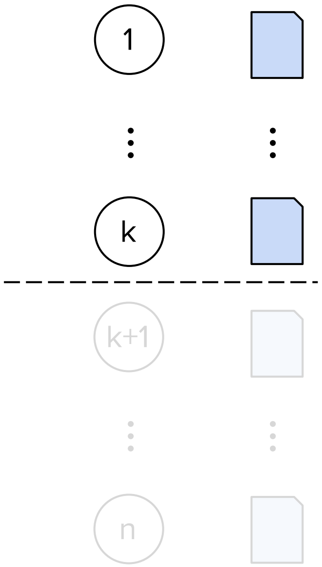
$$\mathrm{NDCG@}k = \frac{\mathrm{DCG@}k}{\mathrm{IDCG@}k}$$

$\mathrm{DCG@}k$ if ranking was perfect

Quantify performance of retrieval

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

Ranking



- **N**ormalized **D**iscounted **C**umulative **G**ain at **k** (**NDCG@k**)
- **R**eciprocal **R**ank at **k** (**RR@k**)

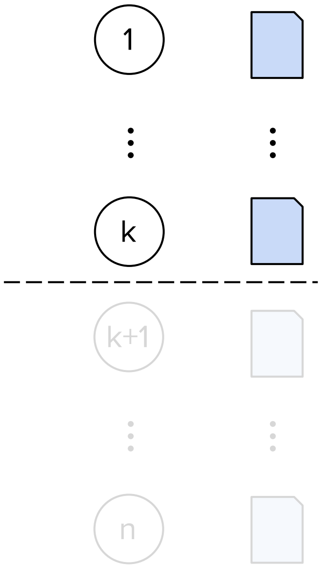
$$\mathrm{RR} = \frac{1}{\mathrm{rank}}$$

rank of the first relevant chunk

Quantify performance of retrieval

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

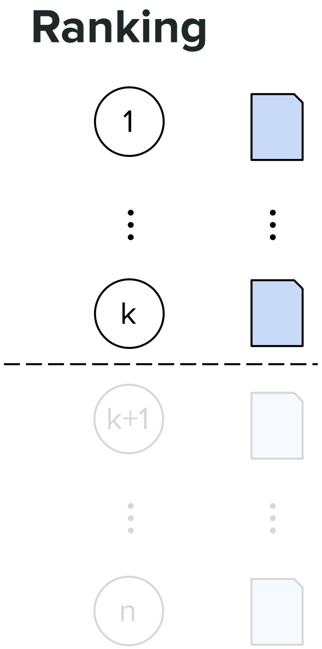
Ranking



- **N**ormalized **D**iscounted **C**umulative **G**ain at **k** (**NDCG@k**)
- **R**eciprocal **R**ank at **k** (**RR@k**)
- **Recall at k**

Quantify performance of retrieval

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

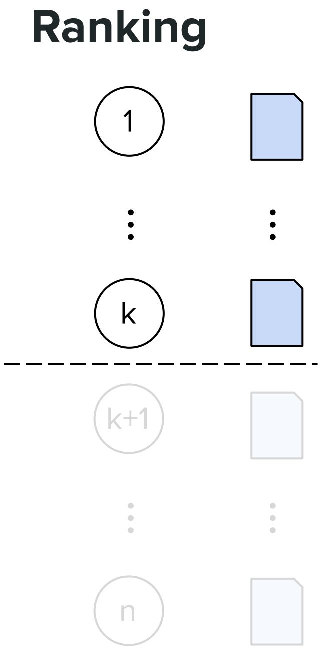


- **N**ormalized **D**iscounted **C**umulative **G**ain at **k** (**NDCG@k**)
- **R**eciprocal **R**ank at **k** (**RR@k**)
- **Recall at k**

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

Quantify performance of retrieval

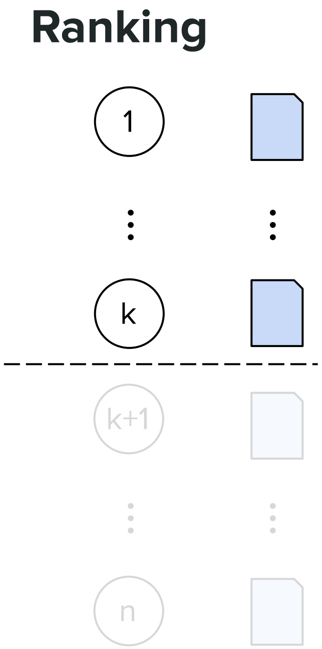
Setup. Evaluate if **retrieved chunks** are **relevant**



- **N**ormalized **D**iscounted **C**umulative **G**ain at **k** (**NDCG@k**)
- **R**eciprocal **R**ank at **k** (**RR@k**)
- **Recall at k**
- **Precision at k**

Quantify performance of retrieval

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



- **N**ormalized **D**iscounted **C**umulative **G**ain at **k** (**NDCG@k**)
- **R**eciprocal **R**ank at **k** (**RR@k**)
- **Recall at k**
- **Precision at k**

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

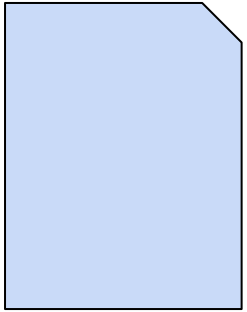
Transformers & Large Language Models

RAG

Tool calling

Agents

Motivation



Unstructured

RAG

Motivation

Unstructured

RAG

ID	Field	…
123	Obs	…
…	…	…

Structured

Idea

ID	Field	…
123	Obs	…
…	…	…

→

Function

```python
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."

"Tool calling overview", IBM.

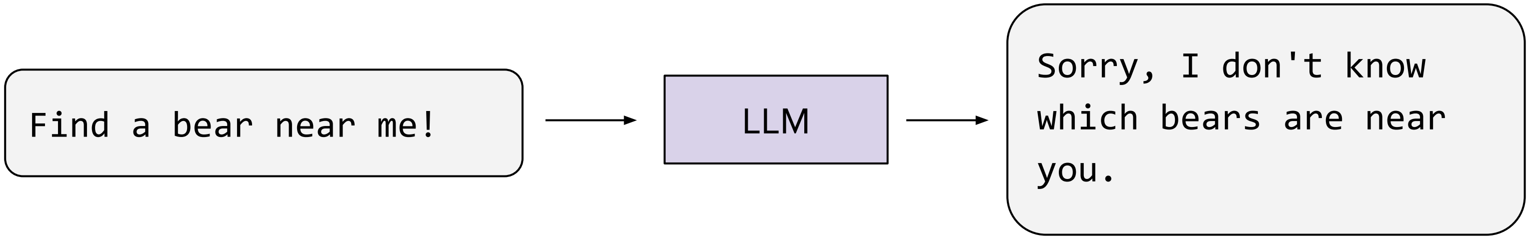
Definition

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

"*Tool calling overview*", *IBM*.

Real-life example

LLM as we've known it so far



Real-life example

LLM as we've known it so far

Find a bear near me! → LLM → Sorry, I don't know which bears are near you.

LLM with tools

<function API>

Find a bear near me! → LLM → ?

Real-life example

LLM as we've known it so far

Find a bear near me! → LLM → Sorry, I don't know which bears are near you.

LLM with tools

<function API>
Find a bear near me! → LLM → ?

Real-life example

find_teddy_bear.py

```python
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}")
```

```python
    # 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!"
    )
```

Real-life example

find_teddy_bear.py

```python
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}")
```

```python
    # 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

```python
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}")
```

```python
    # 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

```python
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}")
```

```python
    # 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

```
<function API>

Find a bear near me!
```

→ LLM →

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

How it works

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

`<function API>`

`Find a bear near me!`

→ LLM →

`location = (37.42,-122.17)`

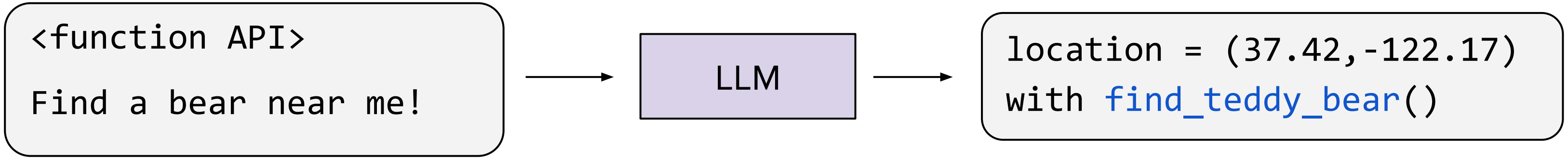
`with find_teddy_bear()`

2. Make **function call**

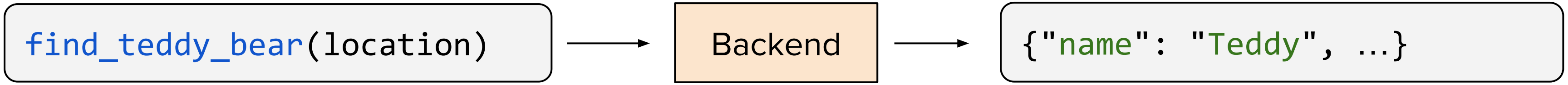
`find_teddy_bear(location)` → Backend → `{"name": "Teddy", ...}`

How it works

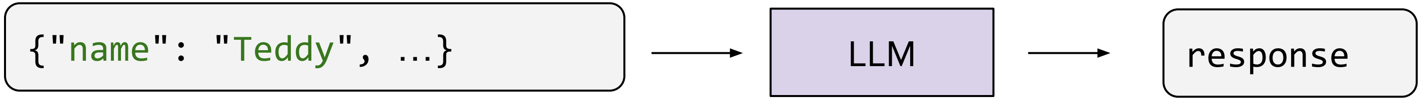
1. Let **LLM** find **argument** for **relevant function** call



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

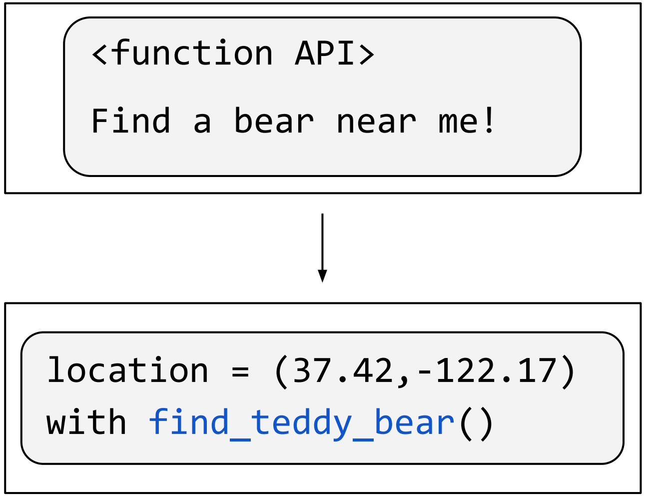
```
<function API>

Find a bear near me!
```

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

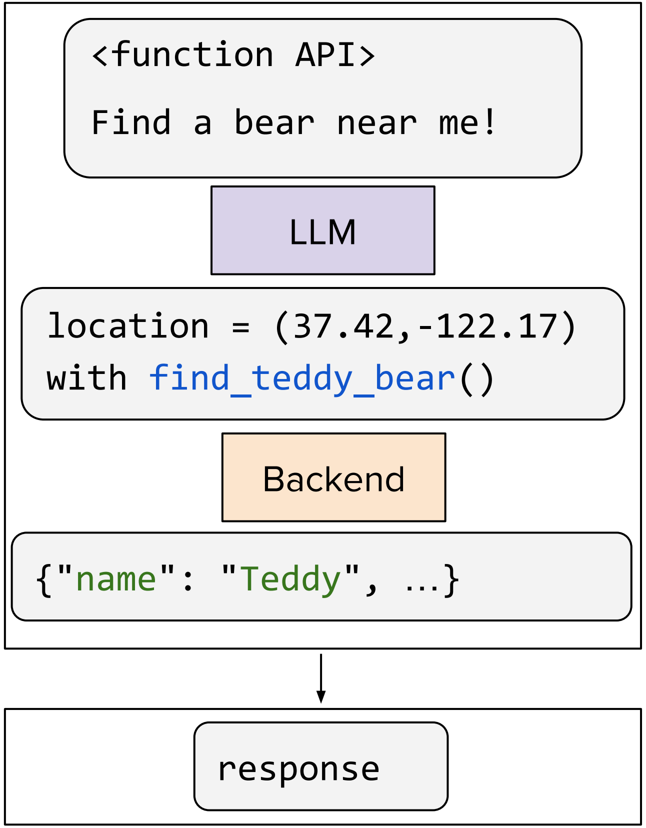
Tool prediction

Teach a model to use a tool

Method 1: via training



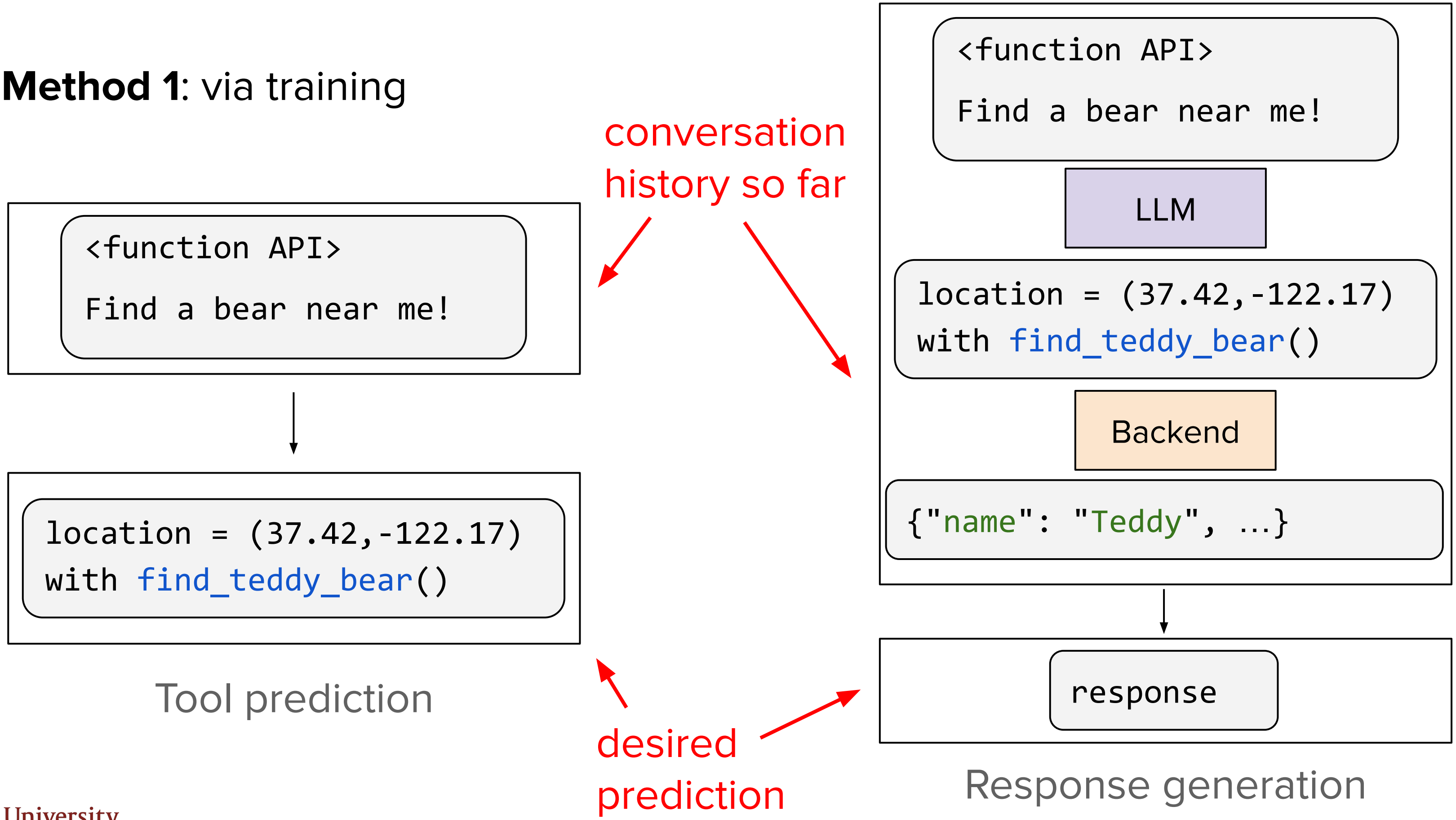
Tool prediction



Response generation

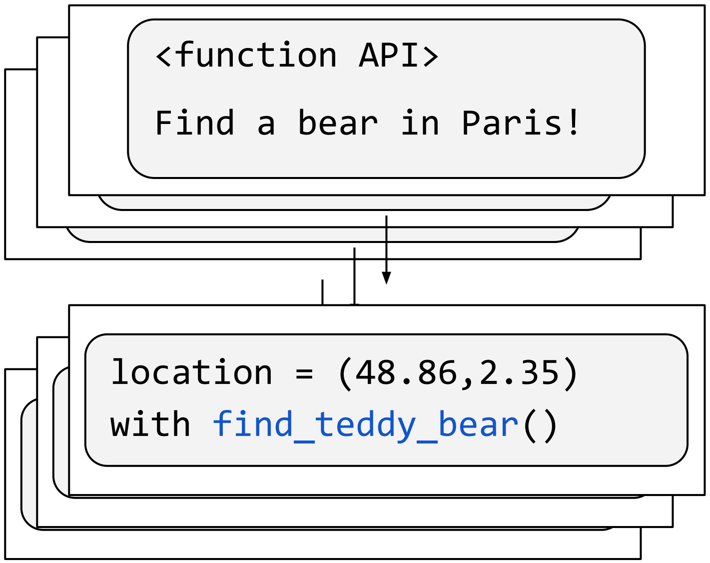
Teach a model to use a tool

Method 1: via training

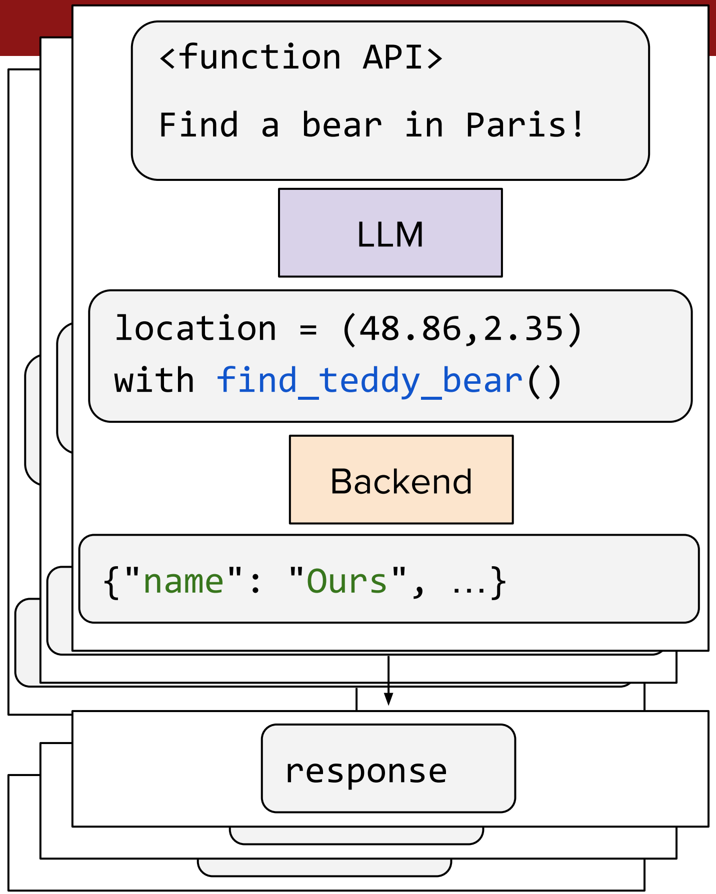


Teach a model to use a tool

Method 1: via training



Tool prediction



Response generation

Teach a model to use a tool

Method 1: via training

Method 2: via prompting

```
<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!
```

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!
```

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

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

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

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

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

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

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

In practice: many tools

~~<function API>~~
~~Find a bear near me!~~
~~→ LLM →~~
~~location = (37.42,-122.17)~~
~~with find_teddy_bear()~~

<**list of** function API**s**>
Find a bear near me!
→ LLM →
location = (37.42,-122.17)
with find_teddy_bear()

Tools summary

Benefits.

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

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.

"Automatic Tool Selection to Reduce Large Language Model Latency", Robert et al., 2024.

Tool selection

Goal. Both reduce latency and improve performance

"*Automatic Tool Selection to Reduce Large Language Model Latency*", *Robert et al., 2024.*

Tool selection

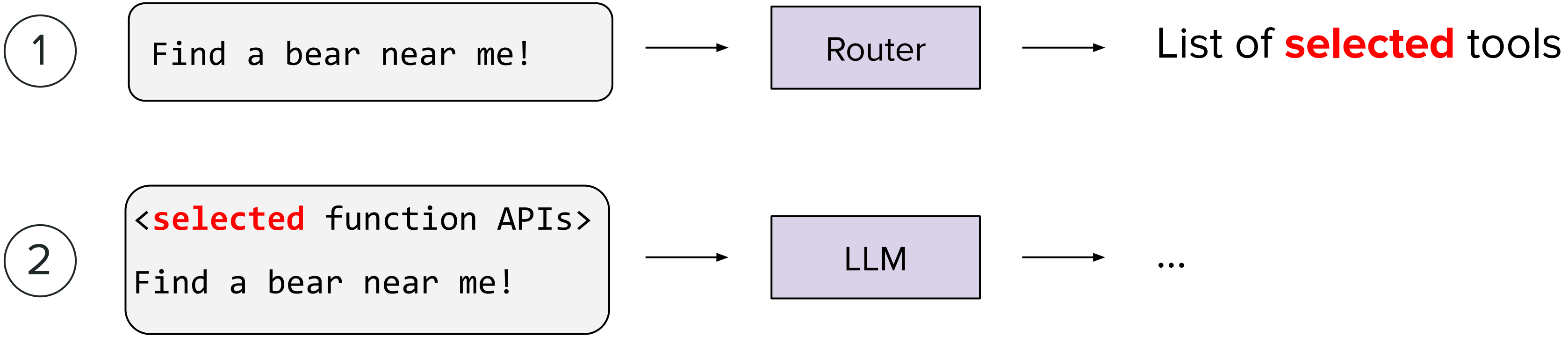
Goal. Both reduce latency and improve performance

1 `Find a bear near me!` → Router → List of **selected** tools

"Automatic Tool Selection to Reduce Large Language Model Latency", Robert et al., 2024.

Tool selection

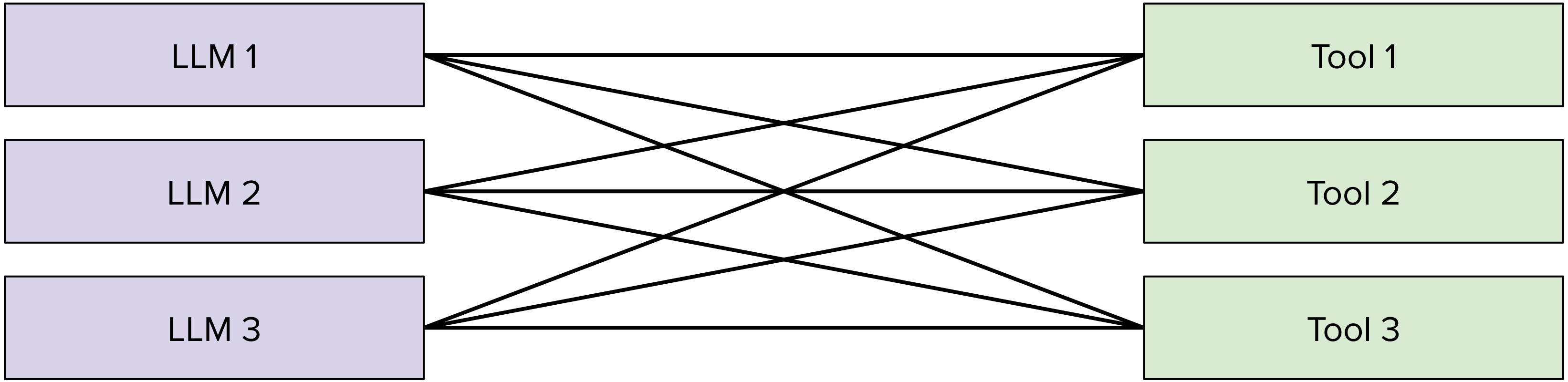
Goal. Both reduce latency and improve performance



"Automatic Tool Selection to Reduce Large Language Model Latency", Robert et al., 2024.

Motivation for standardized protocol

Goal. Avoid duplication of tool implementations



Standardization: MCP

MCP = **M**odel **C**ontext **P**rotocol

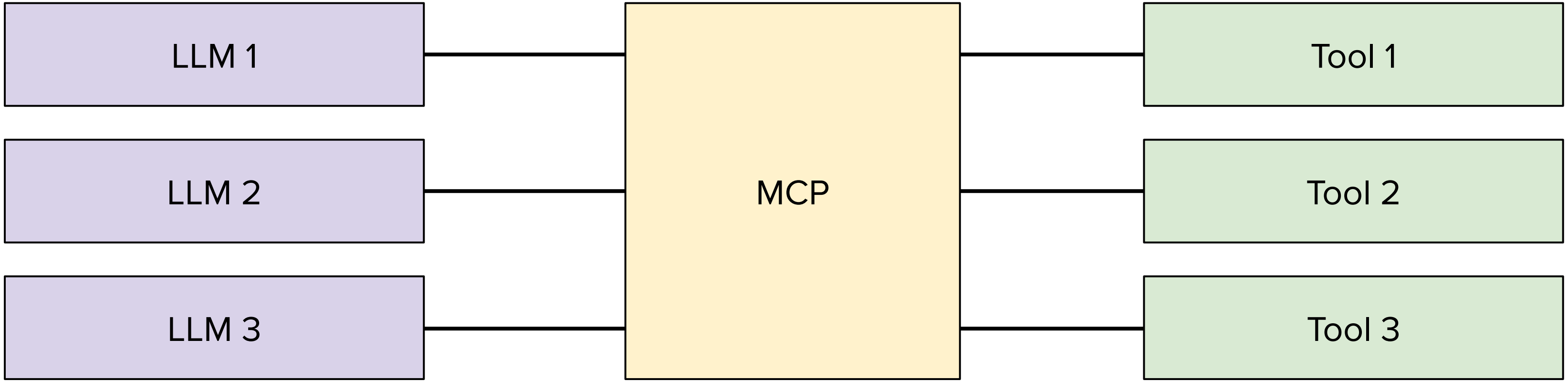
Idea. Connect tools/data to LLMs in a standard way

"*Introducing the Model Context Protocol*", *Anthropic, 2024.*

Standardization: MCP

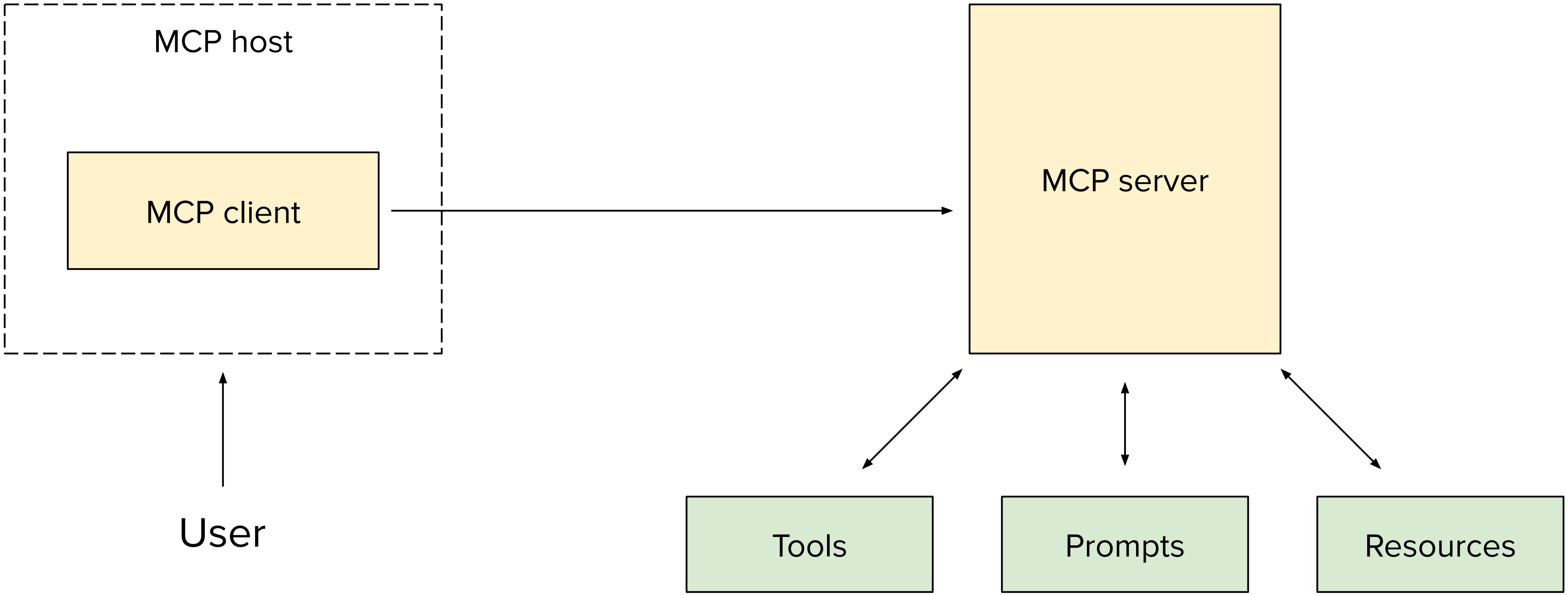
MCP = **M**odel **C**ontext **P**rotocol

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



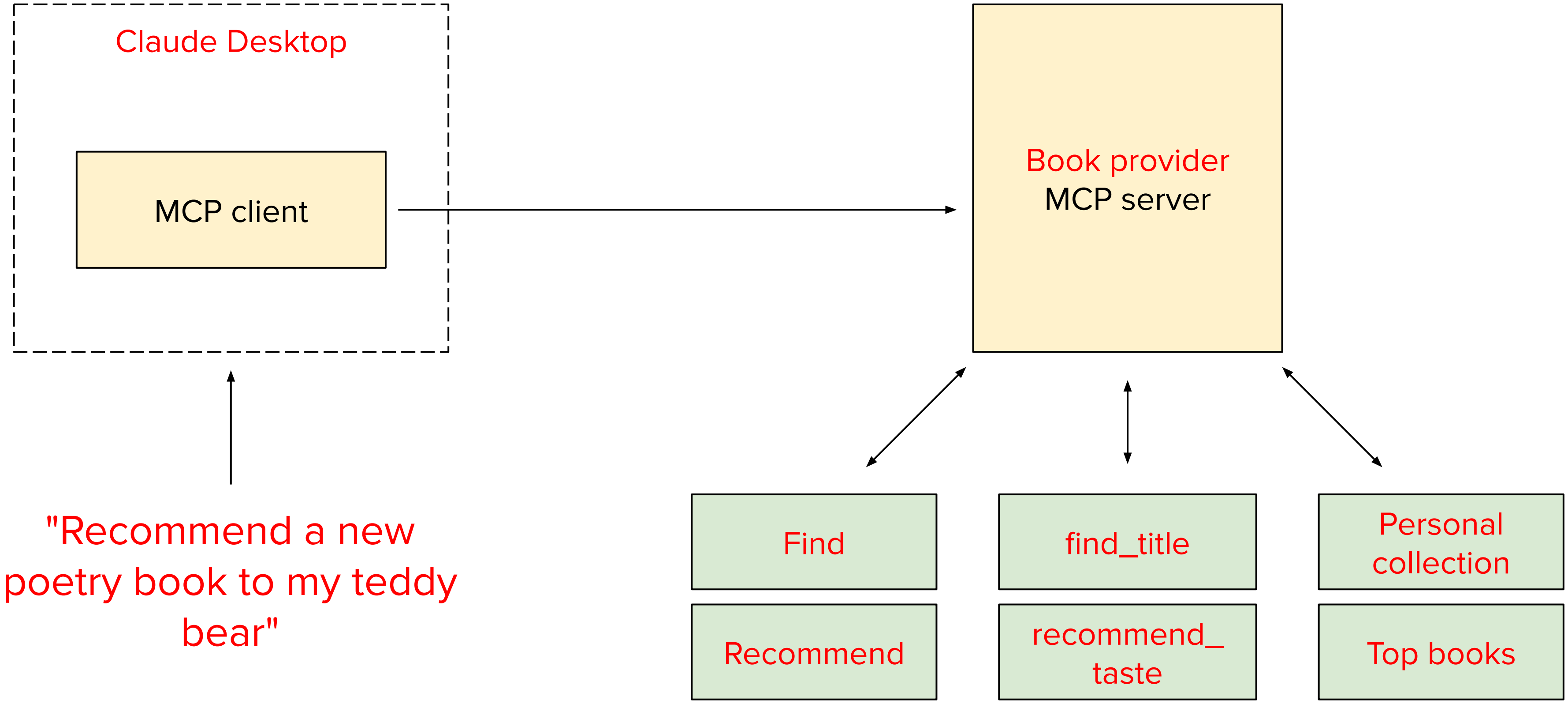
"Introducing the Model Context Protocol", Anthropic, 2024.

Standardization: MCP



"About MCP > Architecture overview", Anthropic, 2024.

Standardization: MCP



"About MCP > Architecture overview", Anthropic, 2024.

Transformers & Large Language Models

RAG

Tool calling

Agents

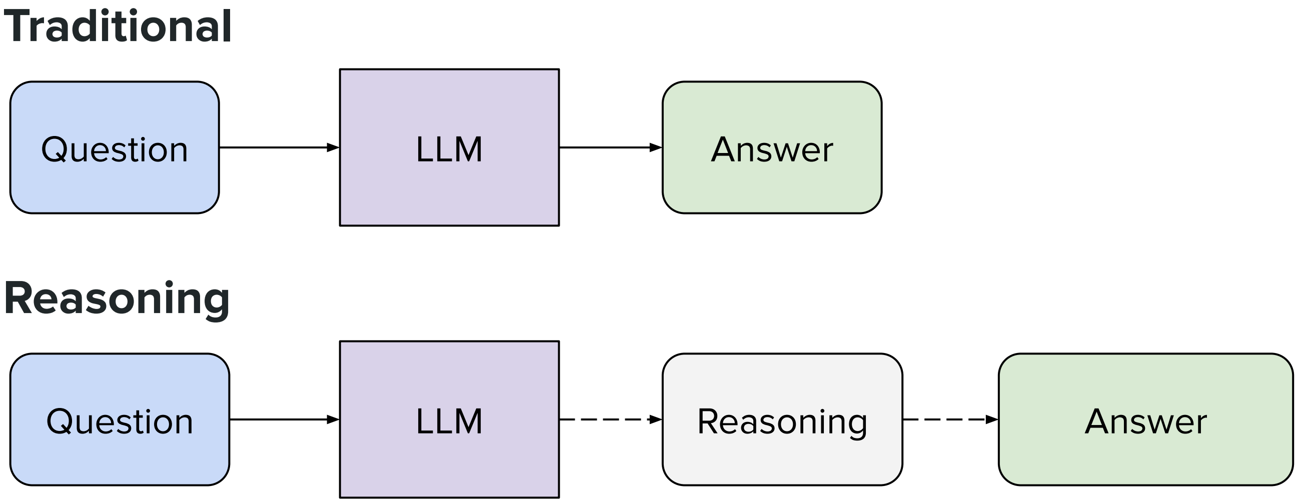
Definition

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

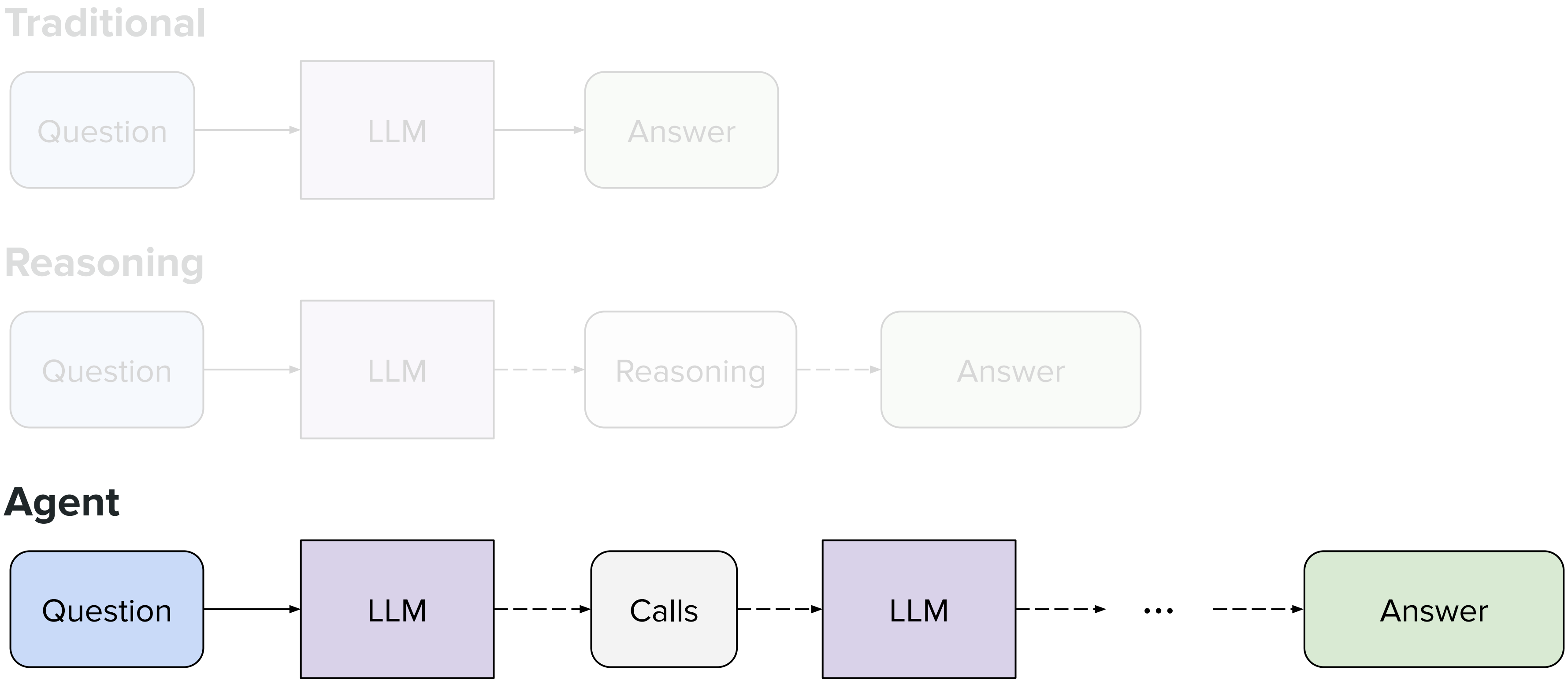
Definition

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

High-level idea

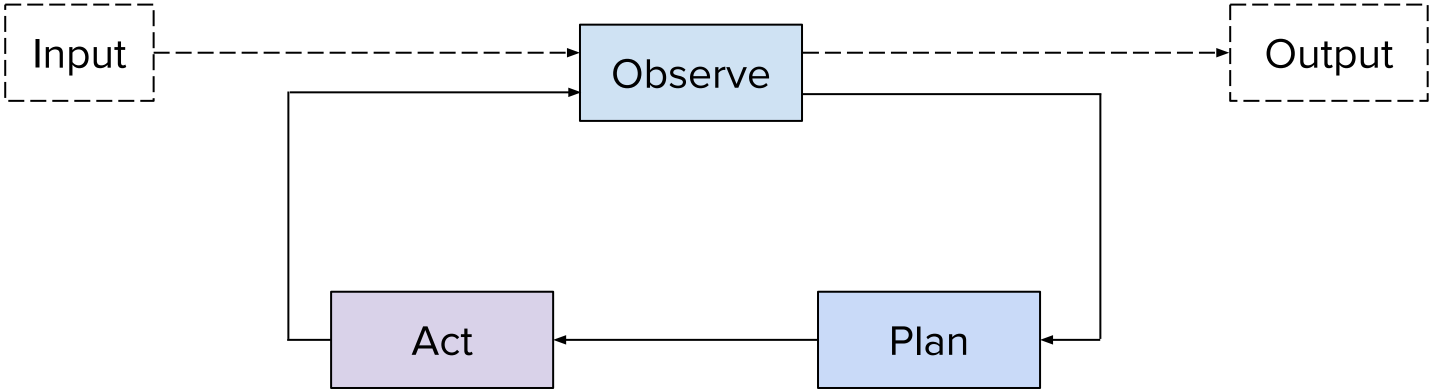


High-level idea



Overview

ReAct = **Re**ason + **Act**



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

ReAct in action

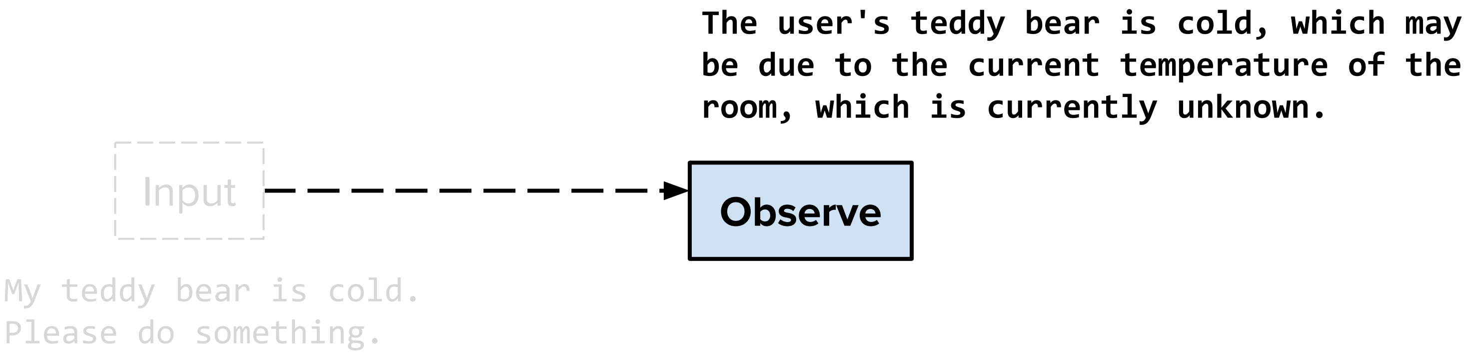
Input

```
My teddy bear is cold.
Please do something.
```

Examples:

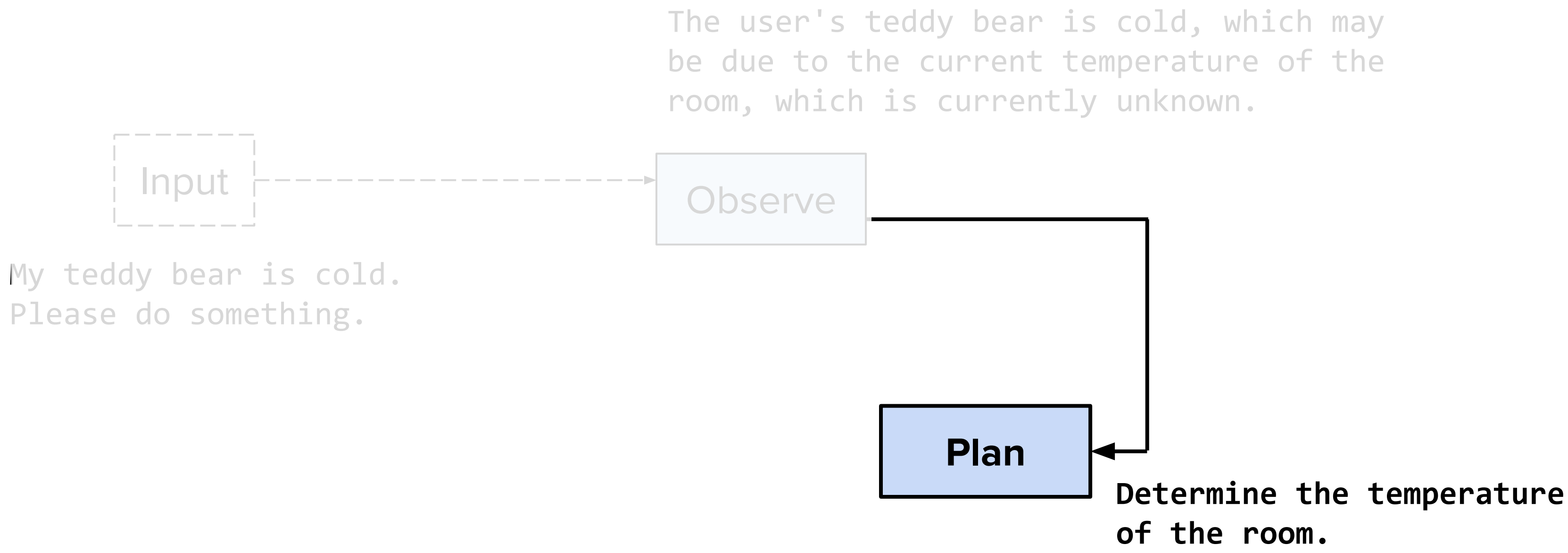
- Manually entered (e.g. user question)
- External event (e.g. metric going beyond a threshold)

ReAct in action



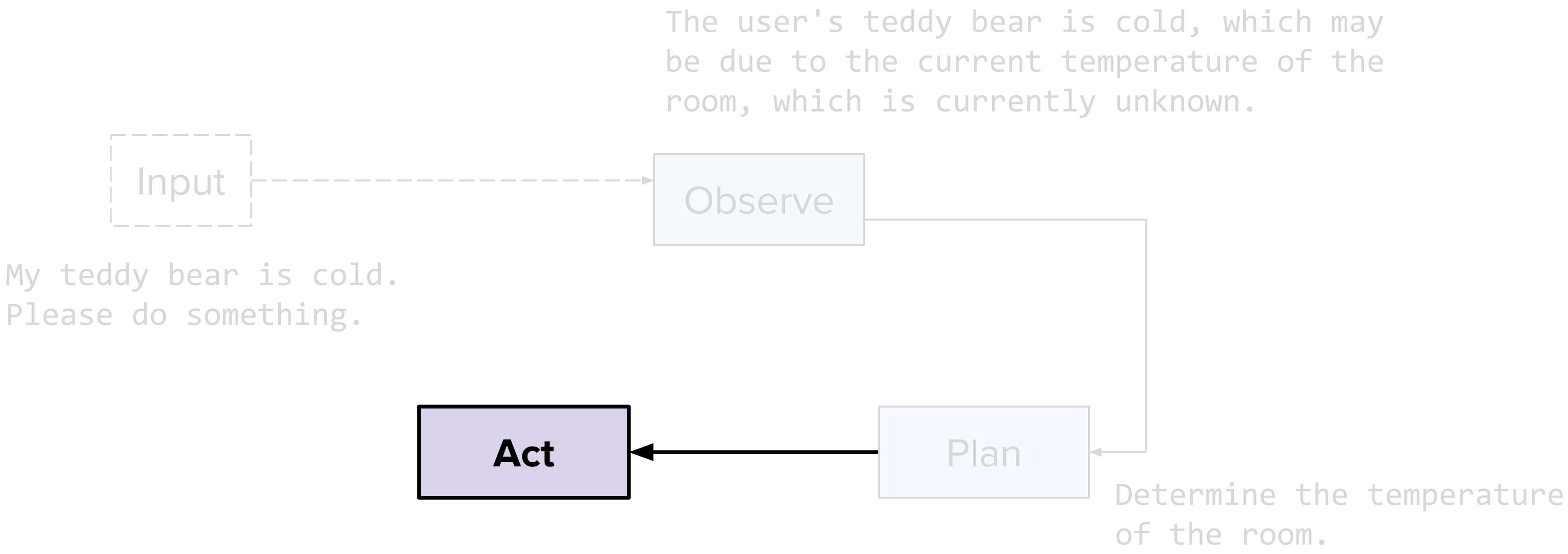
- **Synthesize** previous actions + explicitly **state** what is **currently known** including own knowledge
- **Reasoning**-heavy step to figure out **what is needed**

ReAct in action



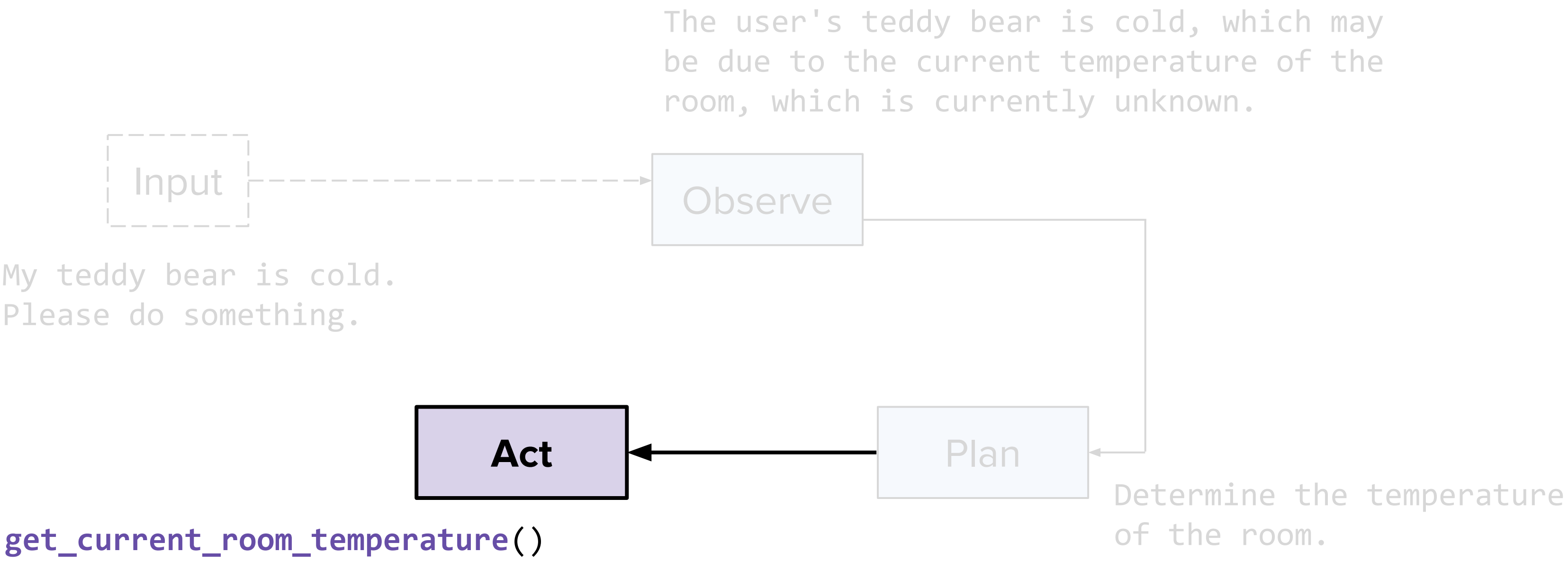
Detail **what tasks** need to be accomplished and **what tools** to call

ReAct in action



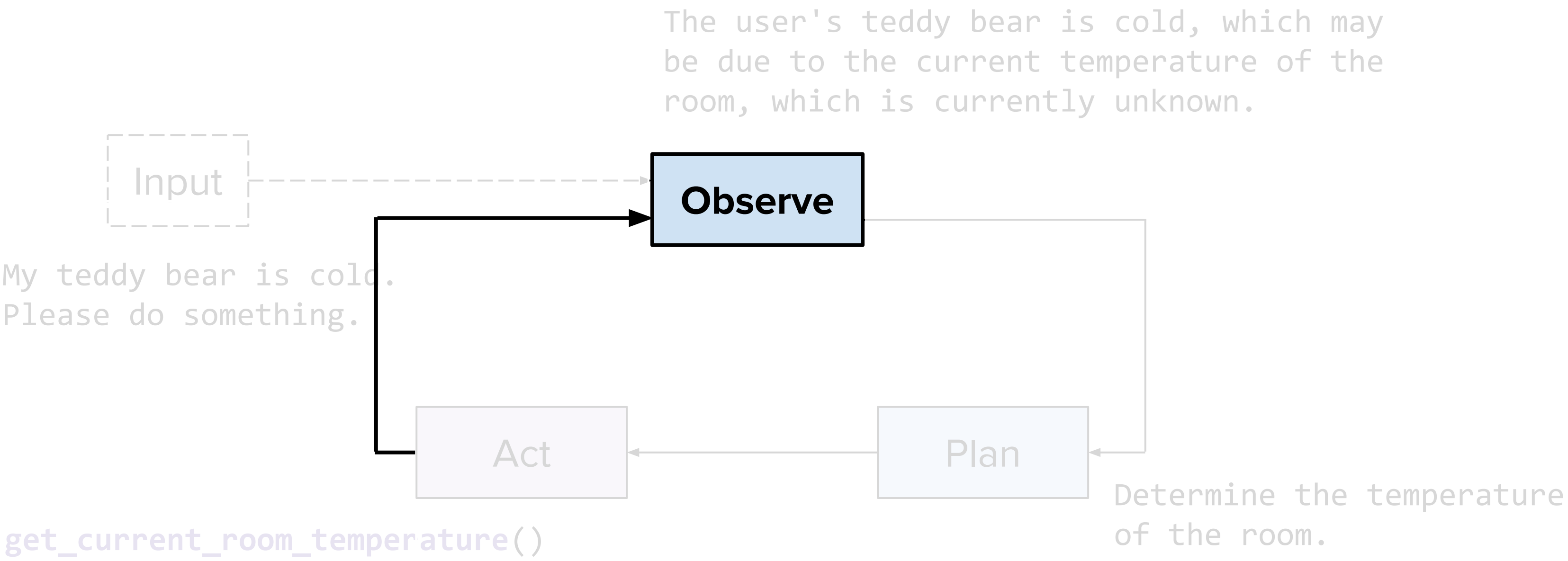
- Perform an action via an **API**
- **Look** for info in a **database** of documents

ReAct in action

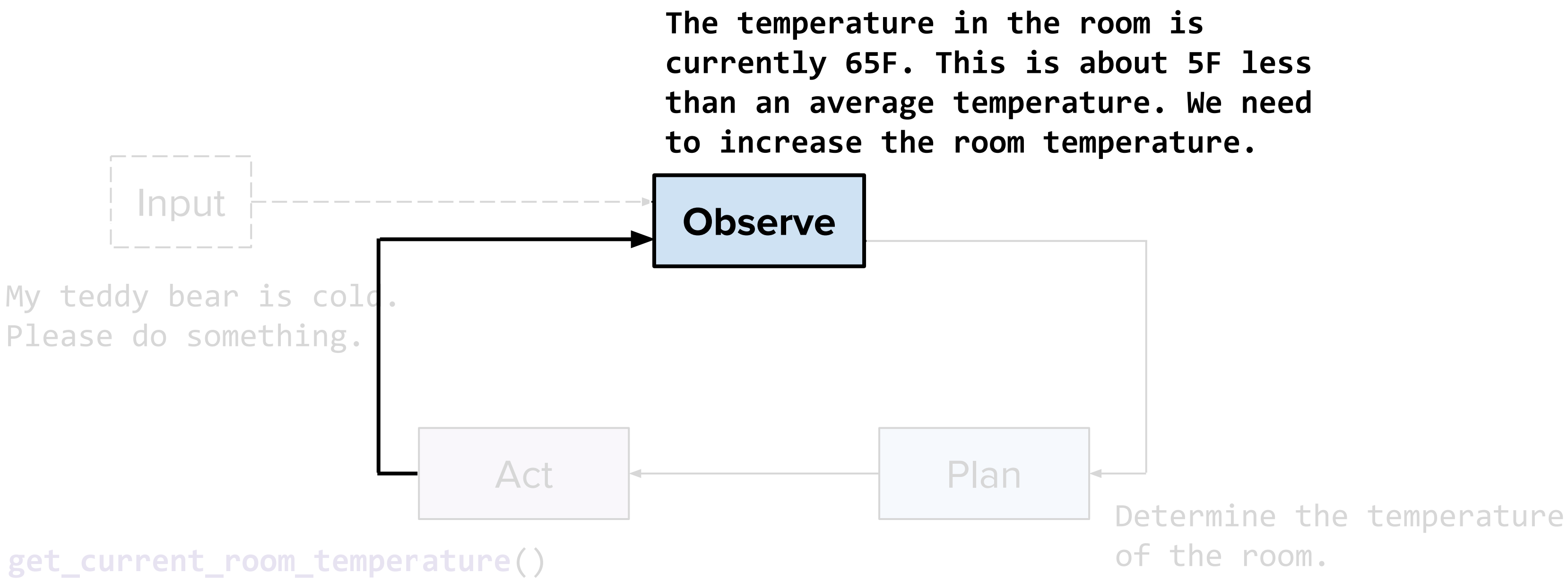


- Perform an action via an **API**
- **Look** for info in a **database** of documents

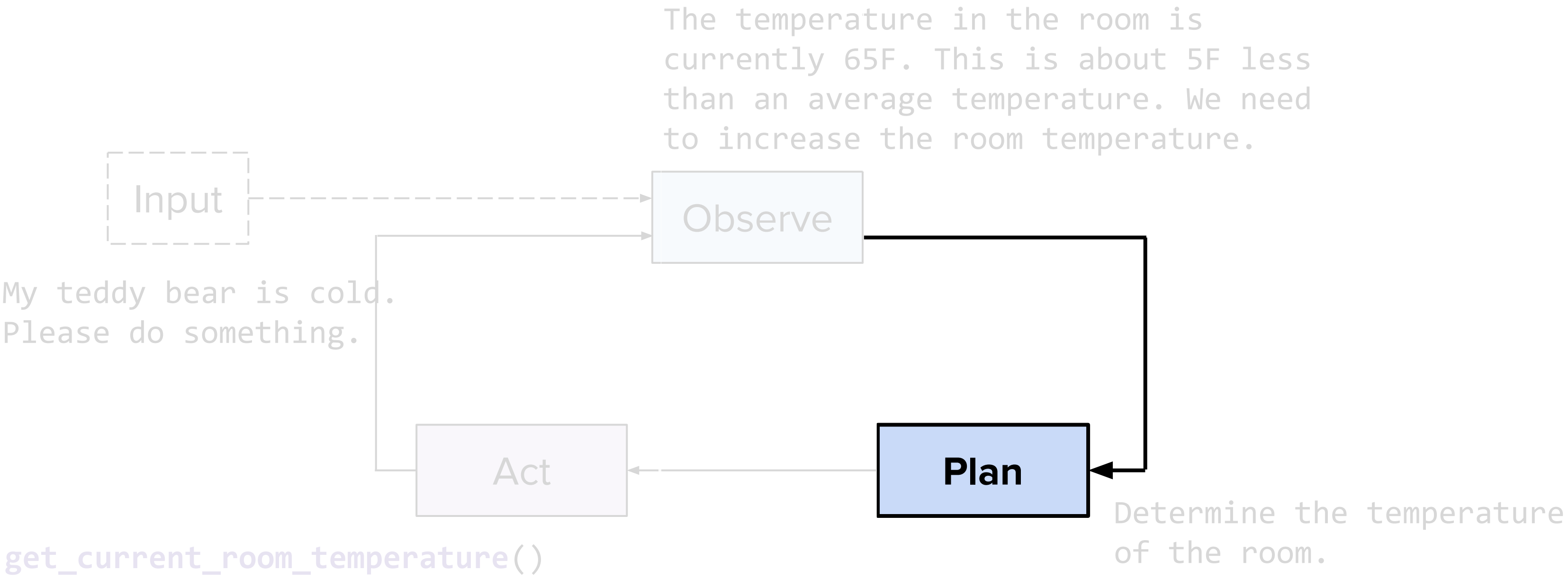
ReAct in action



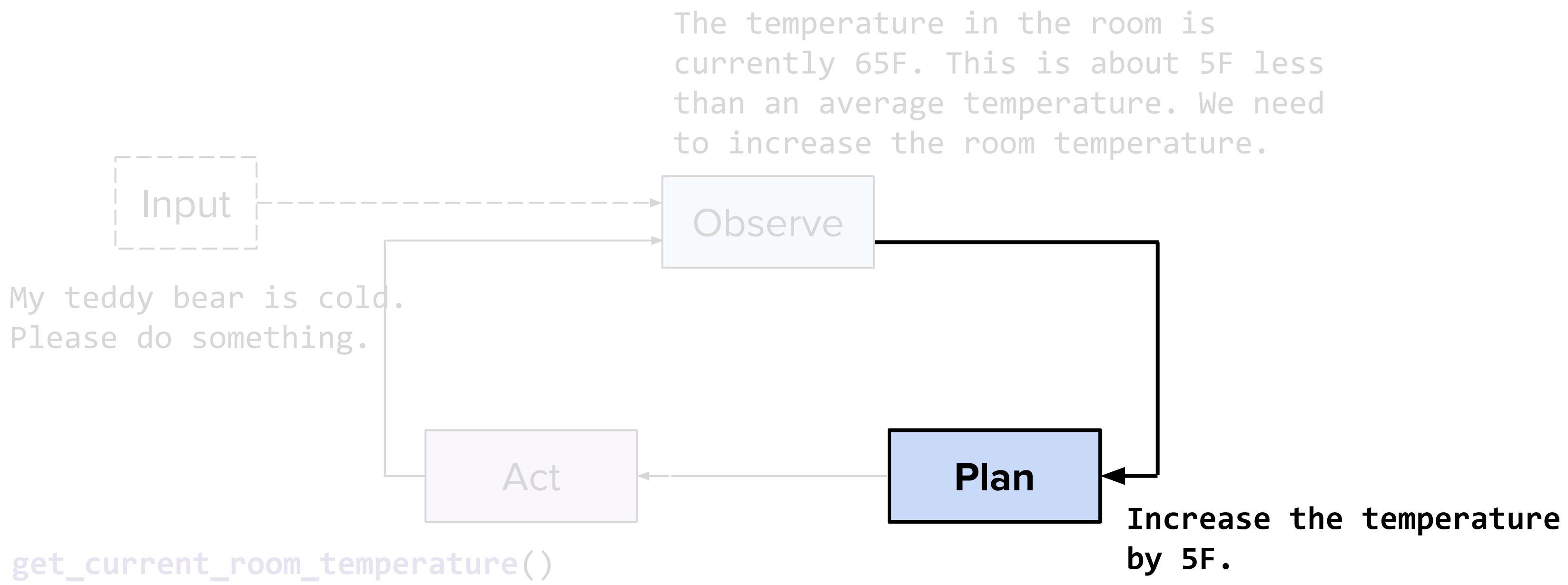
ReAct in action



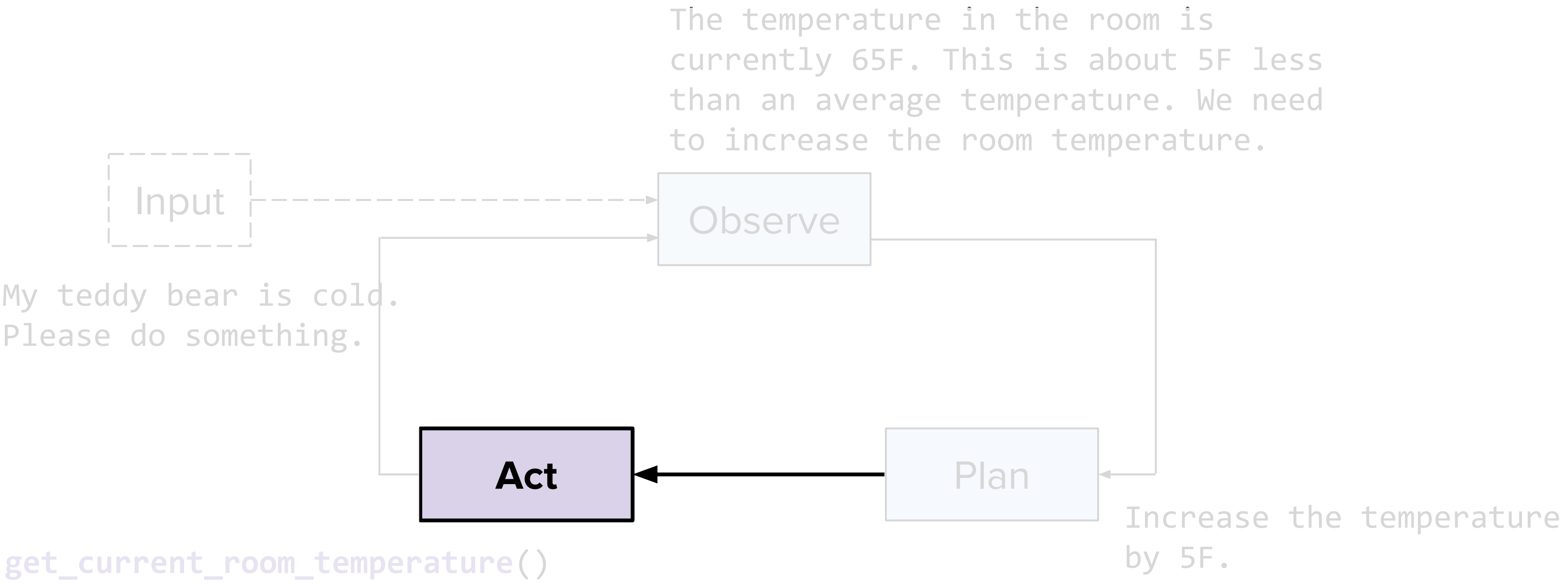
ReAct in action



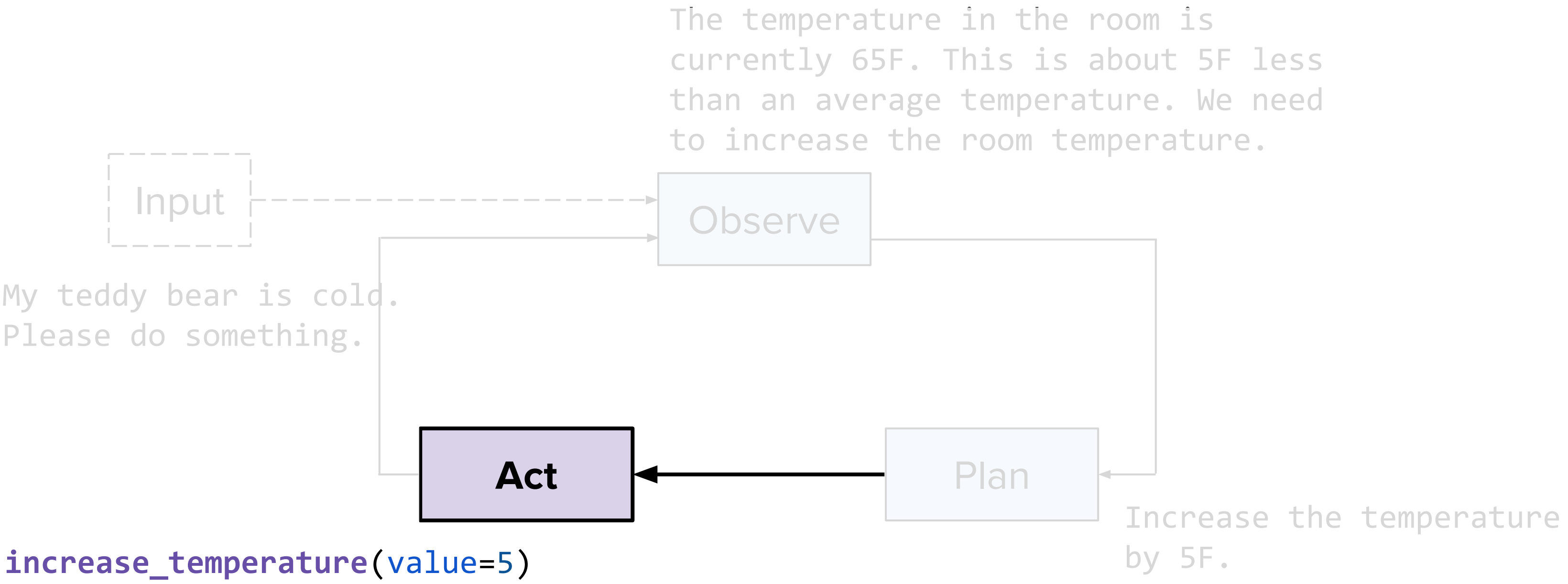
ReAct in action



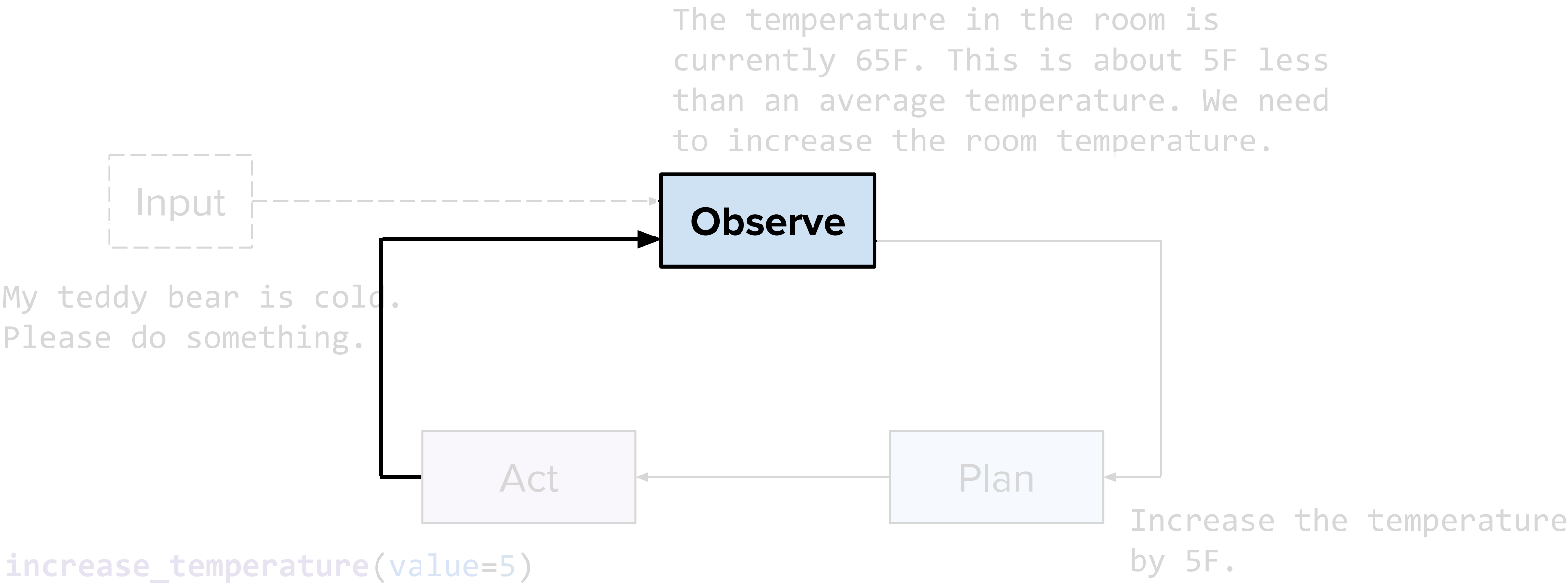
ReAct in action



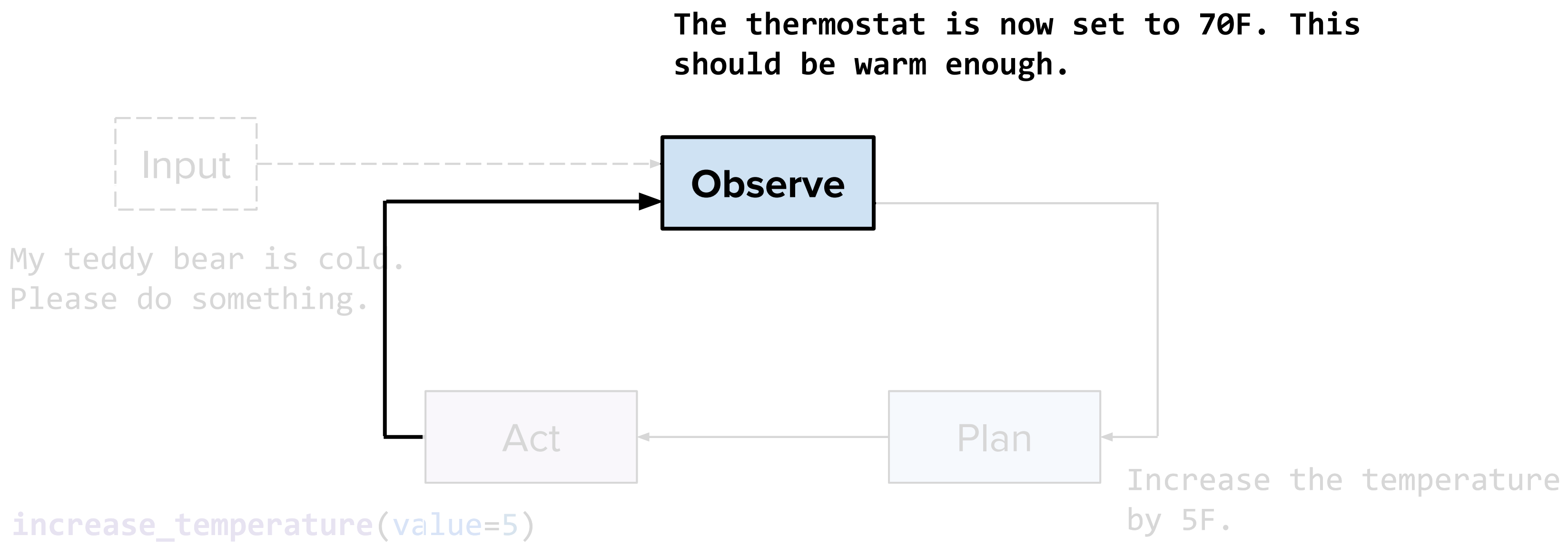
ReAct in action



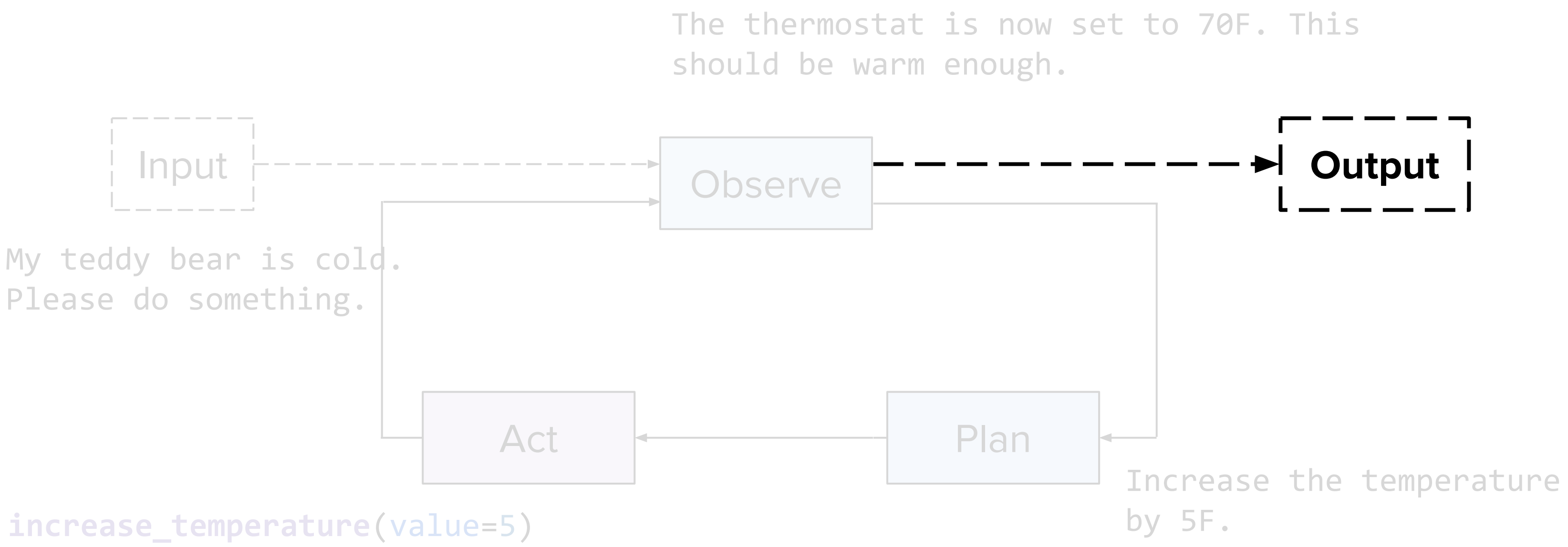
ReAct in action



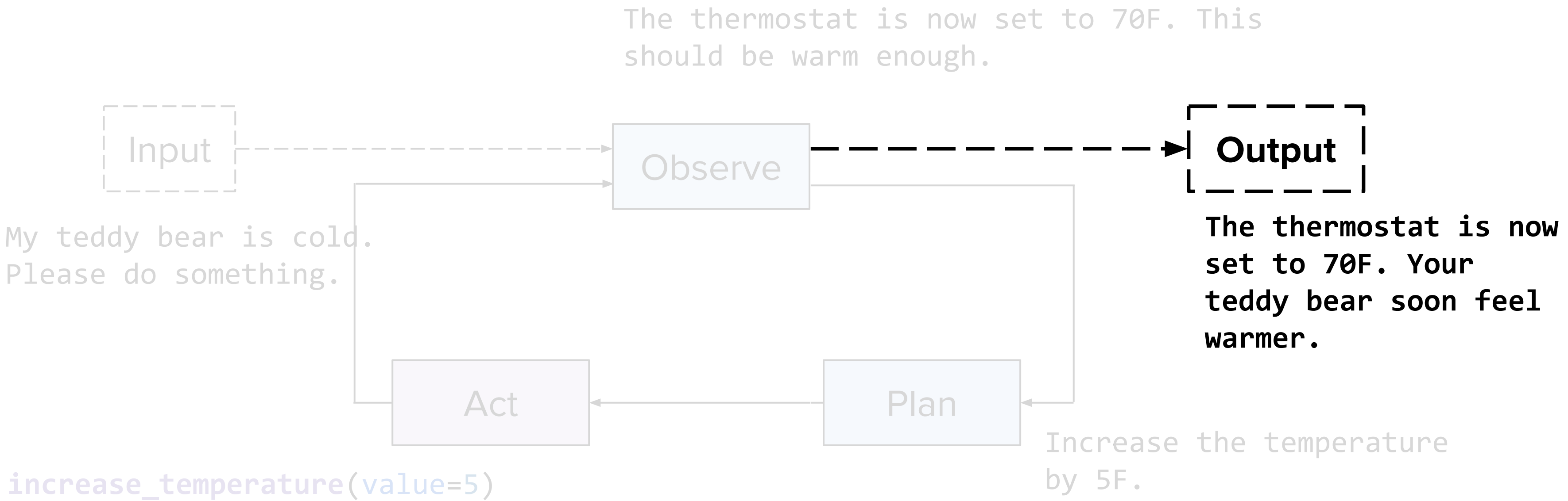
ReAct in action



ReAct in action



ReAct in action



"Agentic" view

`My teddy bear is cold. Please do something.`

`The thermostat is now set to 70F. Your teddy bear soon feel warmer.`

There can be agents for many things

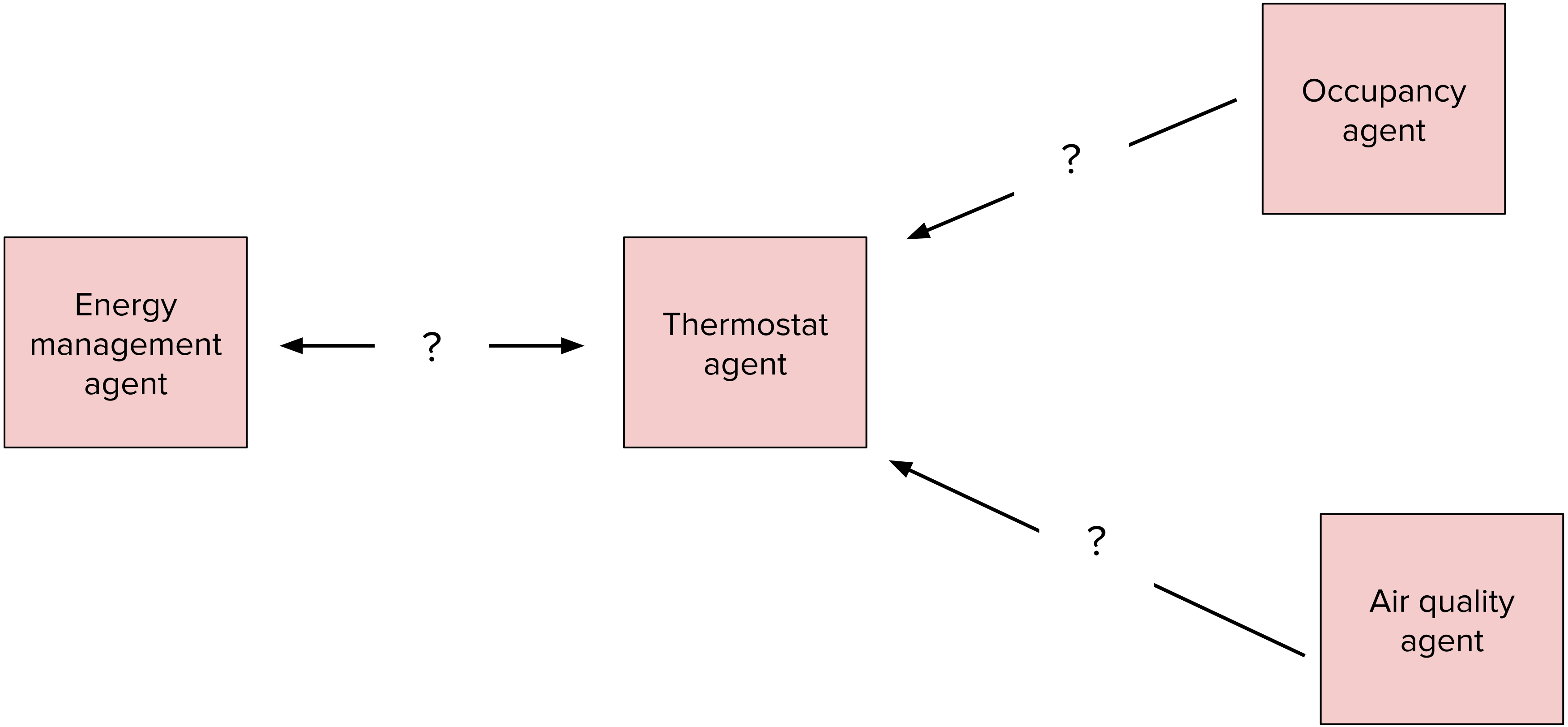
Occupancy agent

Energy management agent

Thermostat agent

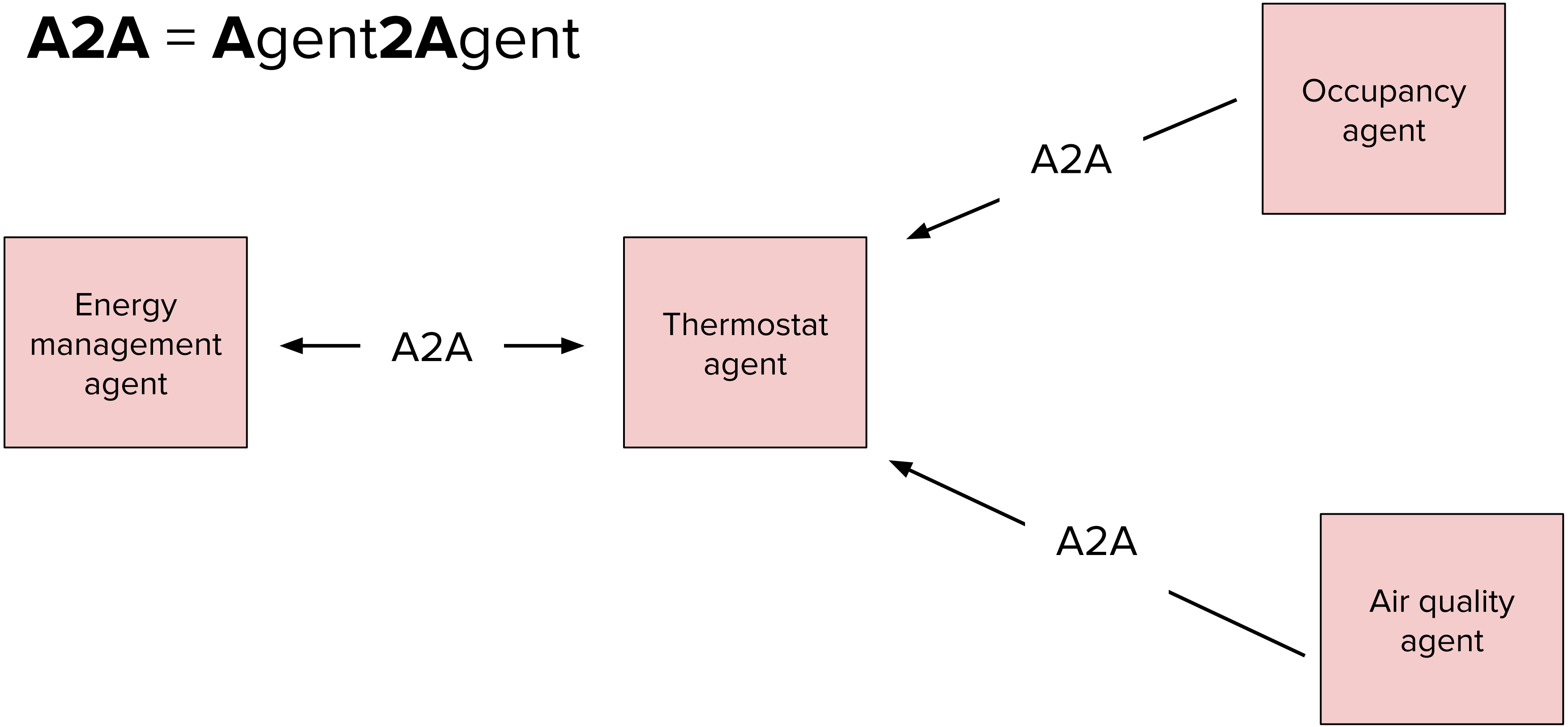
Air quality agent

How can agents communicate with each other?



Standardization: A2A

A2A = **A**gent**2A**gent



"Announcing the Agent2Agent Protocol (A2A)", Google, 2025.

Standardization: A2A

Thermostat agent

=

AgentSkill

```
id='optimize_energy'
description='Reduces energy consumption'
examples=['Save energy overnight']
```

AgentCard

```
name='thermostat agent'
url='http://path.to.agent'
version='1.0.0'
skills=[
 optimize_energy,
 prepare_home,
]
```

AgentExecutor

```
async def execute(self, ...):
  # Logic.

async def cancel(self, ...):
  # Logic.
```


"*Agent2Agent (A2A) Protocol Official Specification*", *Google, 2025.*

Safety

Risks.

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

"*ToolSword: Unveiling Safety Issues of Large Language Models in Tool Learning Across Three Stages*", *Ye et al., 2024.*

Safety

Risks.

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

Remediations.

- Training steps
- Inference safeguards
- Benchmarks, e.g. Agent-SafetyBench

"*Towards Tool Use Alignment of Large Language Models*", *Chen et al., 2024.*

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

"Disrupting the first reported AI-orchestrated cyber espionage campaign", *Anthropic, 2025.*

Closing thoughts

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

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

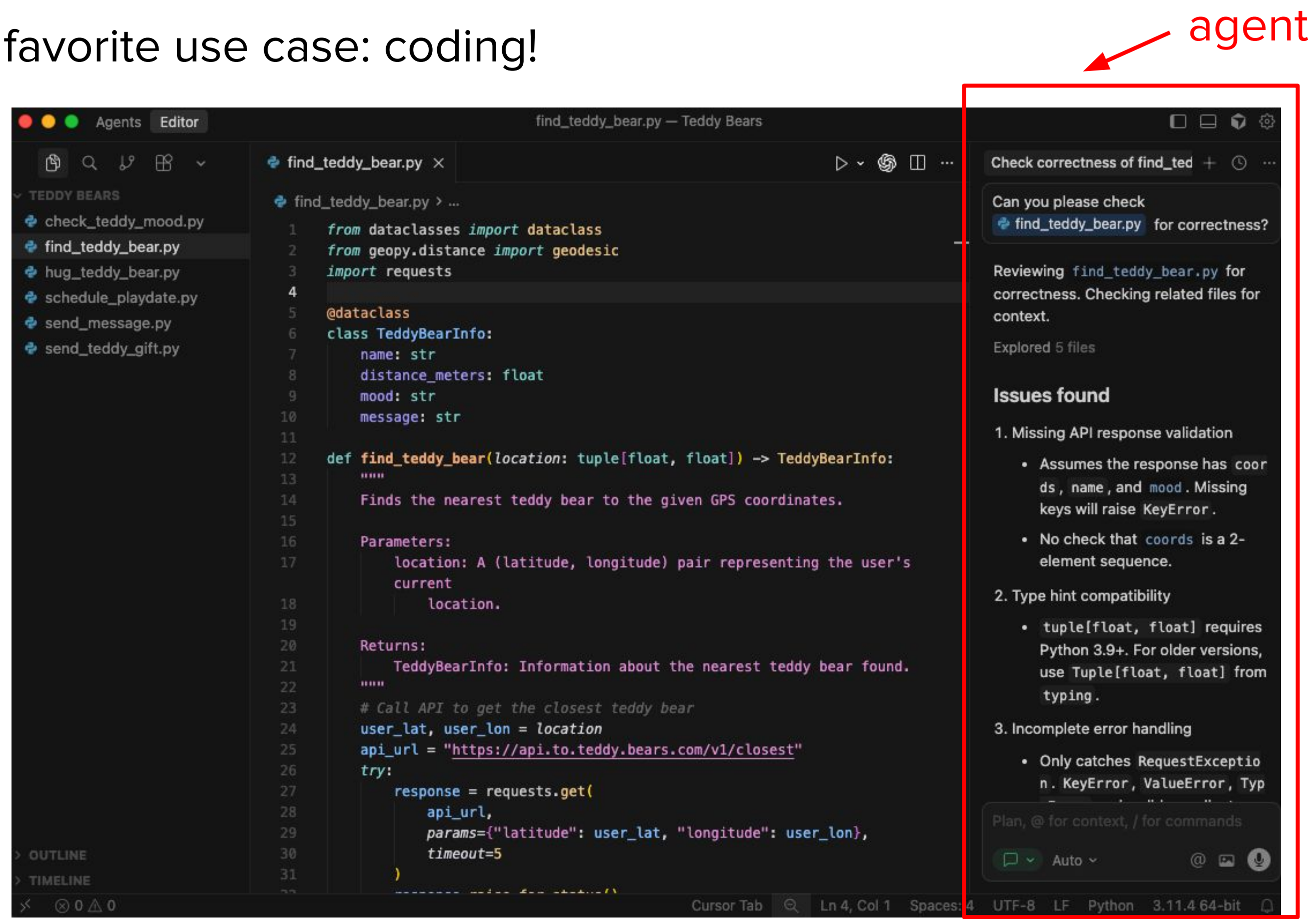
- **Hallucination** is a (big) problem
- **Reasoning** abilities are a bottleneck
    - Finetuning helps, but hard
    - New capabilities are very welcome
- **Evaluation** is challenging
- Good to **start simple**, then **iterate** and **progressively scale up**
- Good to **start** with **capable models**,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



Thank you for your attention!