[at]
alexanderthamm

Workshop: Agentic AI and MCP

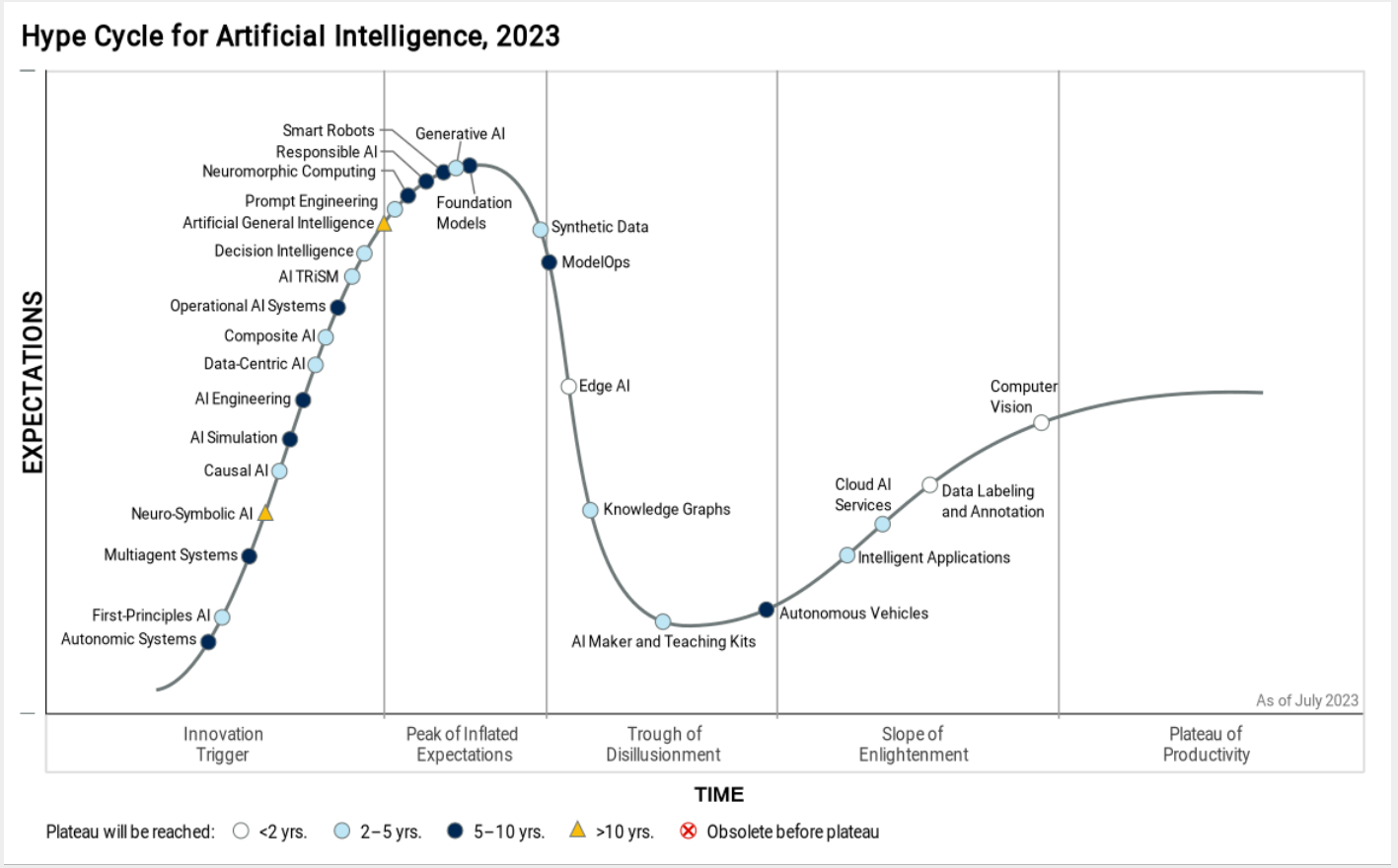
Tore Erdmann

Altana | 21.01.2026

Agenda

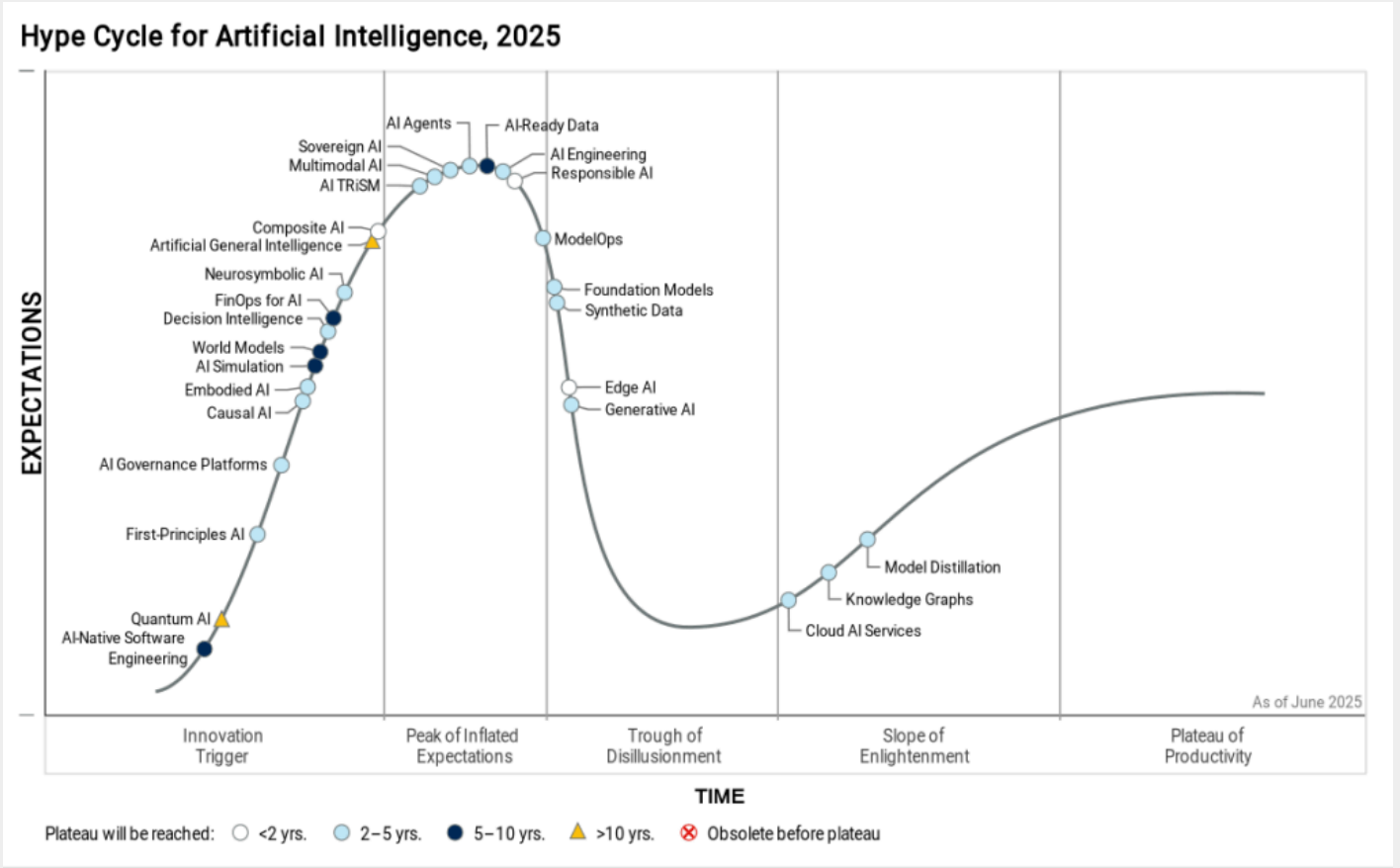
1	**Introduction**	09:00
2	LLM basics	09.15
3	Agents	10.00
4	Lab 1: Deep research	11.00
5	Lunch	12.00
6	MCP	13.00
7	Agents: Advanced Concepts	14.00
8	Lab 2: MCP + advanced agents	15.15
9	Conclusion	16.45

Gartner Hype Cycle



Gartner

Gartner Hype Cycle

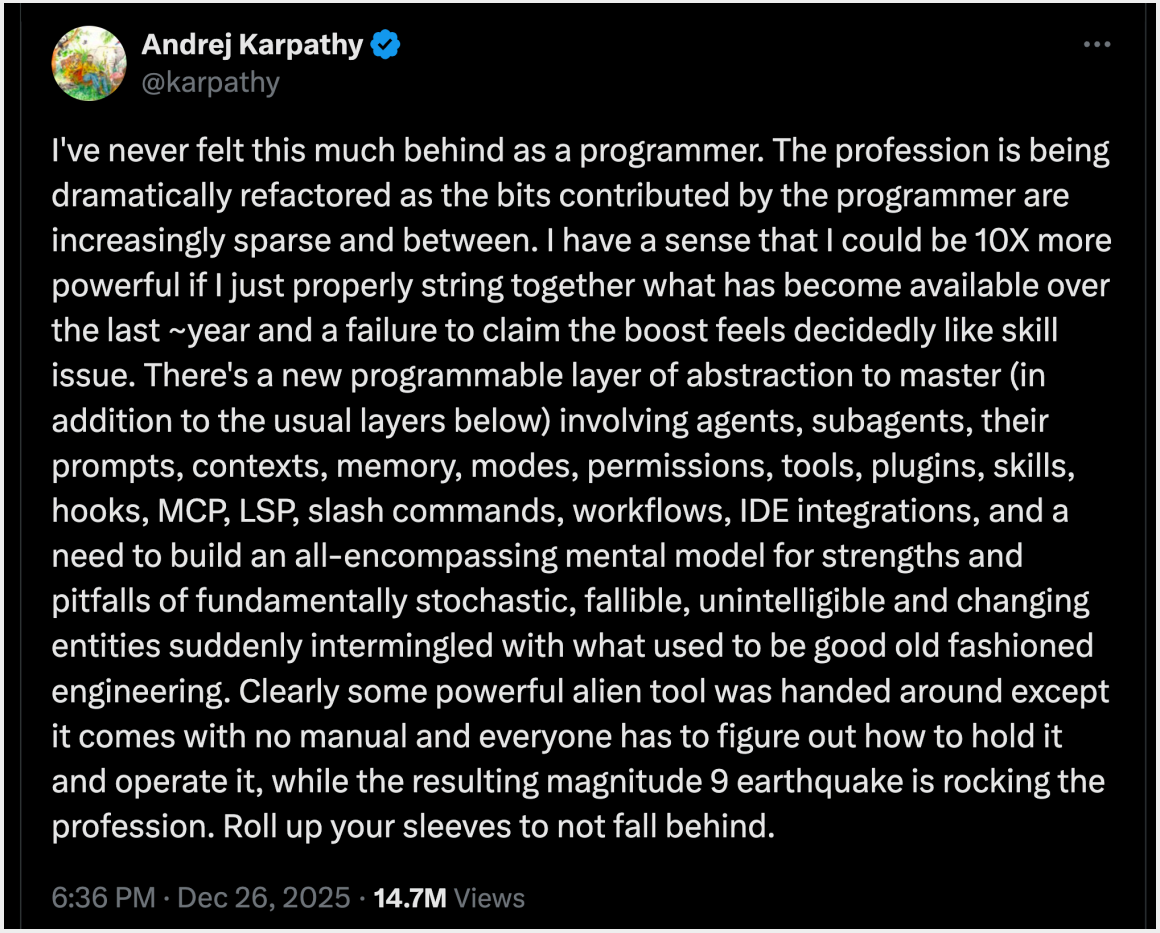


Gartner

Gartner Hype Cycle

“Despite an average spend of $1.9 million on GenAI initiatives in 2024, less than 30% of AI leaders report their CEOs are happy with AI investment return. Low-maturity organizations have trouble identifying suitable use cases and exhibit unrealistic expectations for initiatives. Mature organizations, meanwhile, struggle to find skilled professionals and instill GenAI literacy.”

A. Karpathy on programming

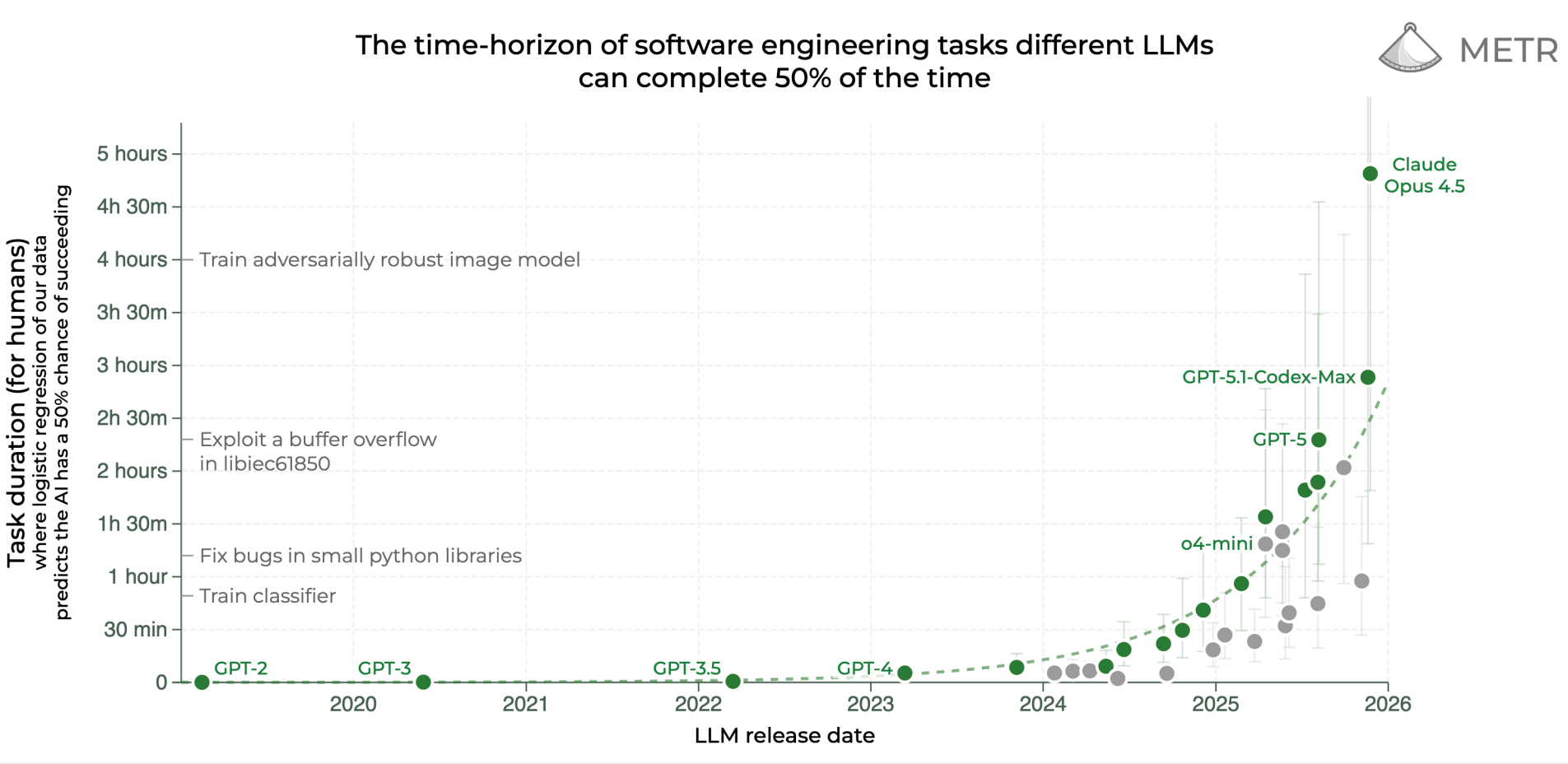


Andrej Karpathy
@karpathy

I've never felt this much behind as a programmer. The profession is being dramatically refactored as the bits contributed by the programmer are increasingly sparse and between. I have a sense that I could be 10X more powerful if I just properly string together what has become available over the last ~year and a failure to claim the boost feels decidedly like skill issue. There's a new programmable layer of abstraction to master (in addition to the usual layers below) involving agents, subagents, their prompts, contexts, memory, modes, permissions, tools, plugins, skills, hooks, MCP, LSP, slash commands, workflows, IDE integrations, and a need to build an all-encompassing mental model for strengths and pitfalls of fundamentally stochastic, fallible, unintelligible and changing entities suddenly intermingled with what used to be good old fashioned engineering. Clearly some powerful alien tool was handed around except it comes with no manual and everyone has to figure out how to hold it and operate it, while the resulting magnitude 9 earthquake is rocking the profession. Roll up your sleeves to not fall behind.

6:36 PM · Dec 26, 2025 · **14.7M** Views

Measuring AI Ability to Complete Long Tasks



Agentic AI and MCP Workshop

- Why are you here?

Agentic AI and MCP Workshop

- Why are you here?
- What experiences do you have with:

Agentic AI and MCP Workshop

- Why are you here?
- What experiences do you have with:
  - LLMs

Agentic AI and MCP Workshop

- Why are you here?
- What experiences do you have with:
  - LLMs
  - Agents

Agentic AI and MCP Workshop

- Why are you here?
- What experiences do you have with:
  - LLMs
  - Agents
  - MCP

Agentic AI and MCP Workshop

- The goals of this workshop

Agentic AI and MCP Workshop

- The goals of this workshop
  - Review LLM basics

Agentic AI and MCP Workshop

- The goals of this workshop
  - Review LLM basics
- LLM Agents concepts

Agentic AI and MCP Workshop

- The goals of this workshop
  - Review LLM basics
- LLM Agents concepts
  - Tool calling + Code execution

Agentic AI and MCP Workshop

- The goals of this workshop
  - Review LLM basics
- LLM Agents concepts
  - Tool calling + Code execution
  - Workflows, Orchestration, Autonomy

Agentic AI and MCP Workshop

- The goals of this workshop
  - Review LLM basics
- LLM Agents concepts
  - Tool calling + Code execution
  - Workflows, Orchestration, Autonomy
- MCP

Agentic AI and MCP Workshop

- The goals of this workshop
  - Review LLM basics
- LLM Agents concepts
  - Tool calling + Code execution
  - Workflows, Orchestration, Autonomy
- MCP
  - Clients + Servers

Agentic AI and MCP Workshop

- What we will do:

Agentic AI and MCP Workshop

- What we will do:
  - Theory / concepts

Agentic AI and MCP Workshop

- What we will do:
  - Theory / concepts
  - Coding exercises

Agentic AI and MCP Workshop

- What we will do:
  - Theory / concepts
  - Coding exercises
  - Lab sessions + try out your own ideas

Setup Hackathon

- Databricks

Setup Hackathon

- Databricks
- Clone the repo at: “https://github.com/terdmann-at/agentic-mcp-hackathon.git”

Setup Hackathon

- Databricks
- Clone the repo at: “https://github.com/terdmann-at/agentic-mcp-hackathon.git”
- Install dependencies from `requirements.txt` on your compute

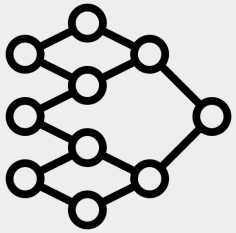
Setup Hackathon

- Databricks
- Clone the repo at: “https://github.com/terdmann-at/agentic-mcp-hackathon.git”
- Install dependencies from `requirements.txt` on your compute
- Turn off AI completions (however do use them if you’re stuck via Cmd+I): Settings/User -> “Autocomplete as you type”, “Automatic Assistant Autocomplete”

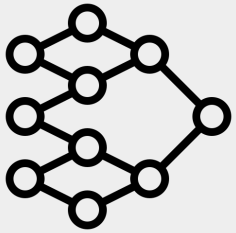
Agenda

1	Introduction	09:00
2	**LLM basics**	09.15
3	Agents	10.00
4	Lab 1: Deep research	11.00
5	Lunch	12.00
6	MCP	13.00
7	Agents: Advanced Concepts	14.00
8	Lab 2: MCP + advanced agents	15.15
9	Conclusion	16.45

How does text-generation with LLMs work?

Prompt	Model	Vocabulary		Probabilities		Sample
I am giving a …		talk	→	**0.90**	→	**talk**
		green	→	0.6		
		chair	→	0.75		
		but	→	0.01		
		…	→	…		

How does text-generation with LLMs work?

Prompt	Model	Vocabulary		Probabilities		Sample
I am giving a talk…		choice	→	0.12		
		about	→	**0.91**	→	**about**
		especially	→	0.43		
		cellar	→	0.01		
		…	→	…		

How does text-generation with LLMs work?

- given some words $w_1, w_2, ..., w_t$, LLMs predict the probability of the next word w_{t+1}

How does text-generation with LLMs work?

- given some words $w_1, w_2, ..., w_t$, LLMs predict the probability of the next word w_{t+1}
- LLMs are a model of:

$$P_\theta(w_{t+1} \mid w_1, w_2, ..., w_t)$$

How does text-generation with LLMs work?

- given some words $w_1, w_2, ..., w_t$, LLMs predict the probability of the next word w_{t+1}
- LLMs are a model of:

$$P_\theta(w_{t+1} \mid w_1, w_2, ..., w_t)$$

- the model weights θ are pre-trained on huge amounts of text

How does text-generation with LLMs work?

- given some words $w_1, w_2, ..., w_t$, LLMs predict the probability of the next word w_{t+1}
- LLMs are a model of:

$$P_\theta(w_{t+1} \mid w_1, w_2, ..., w_t)$$

- the model weights θ are pre-trained on huge amounts of text
- and then trained some more (post-training) to follow instructions

How does text-generation with LLMs work?

- given some words $w_1, w_2, ..., w_t$, LLMs predict the probability of the next word w_{t+1}
- LLMs are a model of:

$$P_\theta(w_{t+1} \mid w_1, w_2, ..., w_t)$$

- the model weights θ are pre-trained on huge amounts of text
- and then trained some more (post-training) to follow instructions
- the words $w_1, ...$ are the “prompt” and allow us to use the model

What is so great about LLMs?

- In-Context Learning

What is so great about LLMs?

- In-Context Learning
- instead of training a model on a dataset, we just write instructions

What is so great about LLMs?

- In-Context Learning
- instead of training a model on a dataset, we just write instructions
- the model can learn from examples in the prompt

What is so great about LLMs?

- In-Context Learning
- instead of training a model on a dataset, we just write instructions
- the model can learn from examples in the prompt
- 0-shot performance / reasoning

What is so great about LLMs?

- In-Context Learning
- instead of training a model on a dataset, we just write instructions
- the model can learn from examples in the prompt
- 0-shot performance / reasoning
- Why does it work?

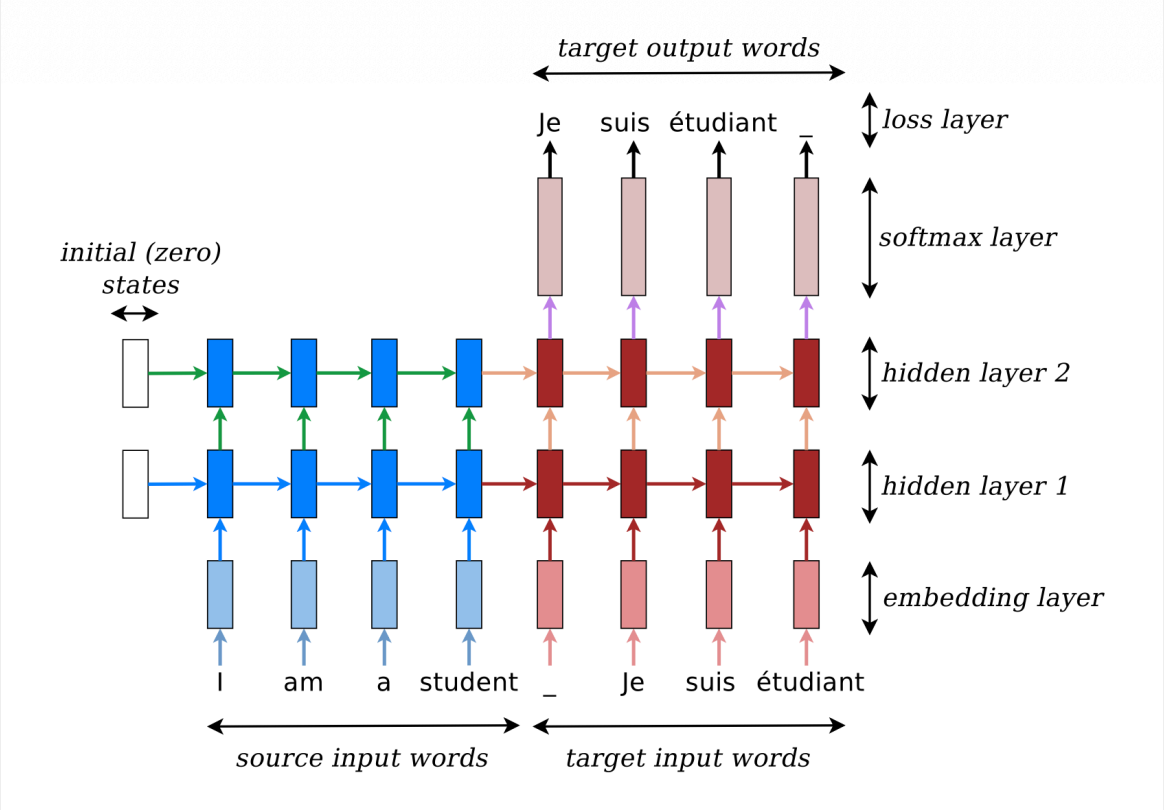
What is so great about LLMs?

- In-Context Learning
- instead of training a model on a dataset, we just write instructions
- the model can learn from examples in the prompt
- 0-shot performance / reasoning
- Why does it work?
- large datasets

What is so great about LLMs?

- In-Context Learning
- instead of training a model on a dataset, we just write instructions
- the model can learn from examples in the prompt
- 0-shot performance / reasoning
- Why does it work?
- large datasets
- attention mechanism

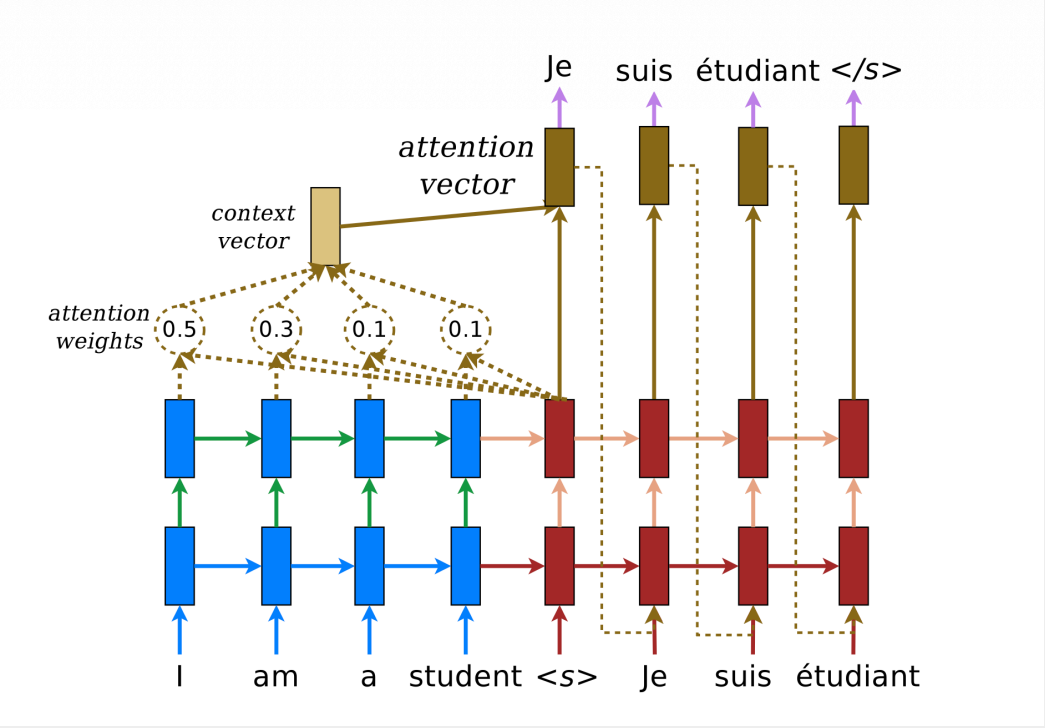
Sequence-to-sequence models



M.-T. Luong. "Neural Machine Translation", PhD Thesis, Stanford CS Dept., 2016

Attention

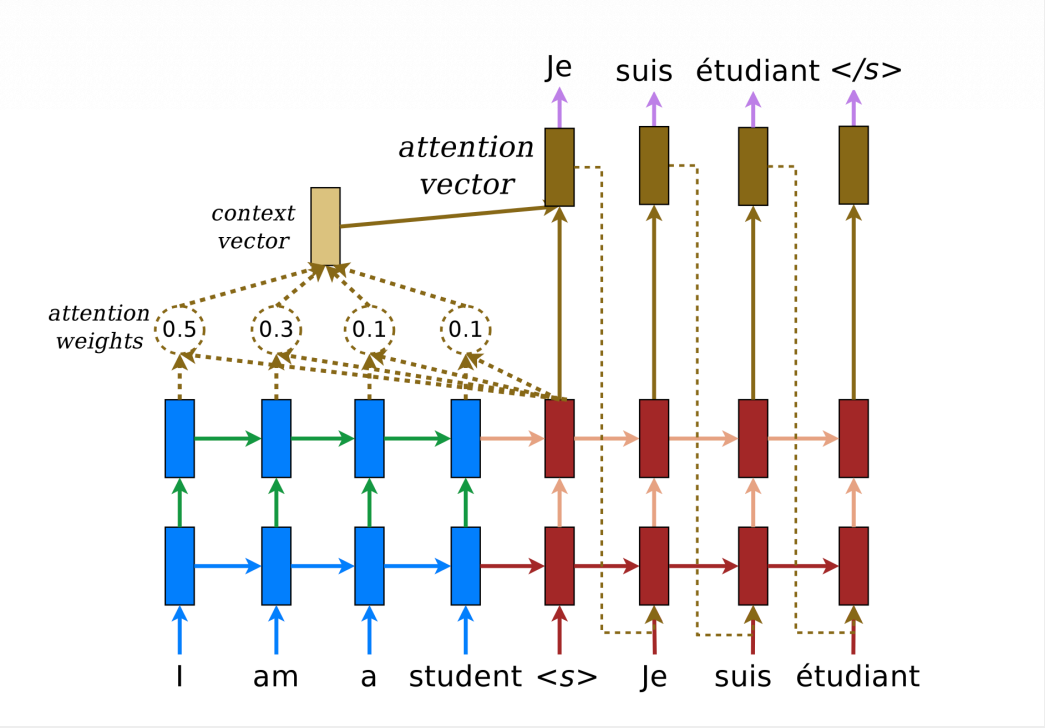
- a seq to seq model: $h_t^d = f_d(h_{t-1}^d, y_{t-1}, c)$



M.-T. Luong. "Neural Machine Translation", PhD Thesis, Stanford CS Dept., 2016

Attention

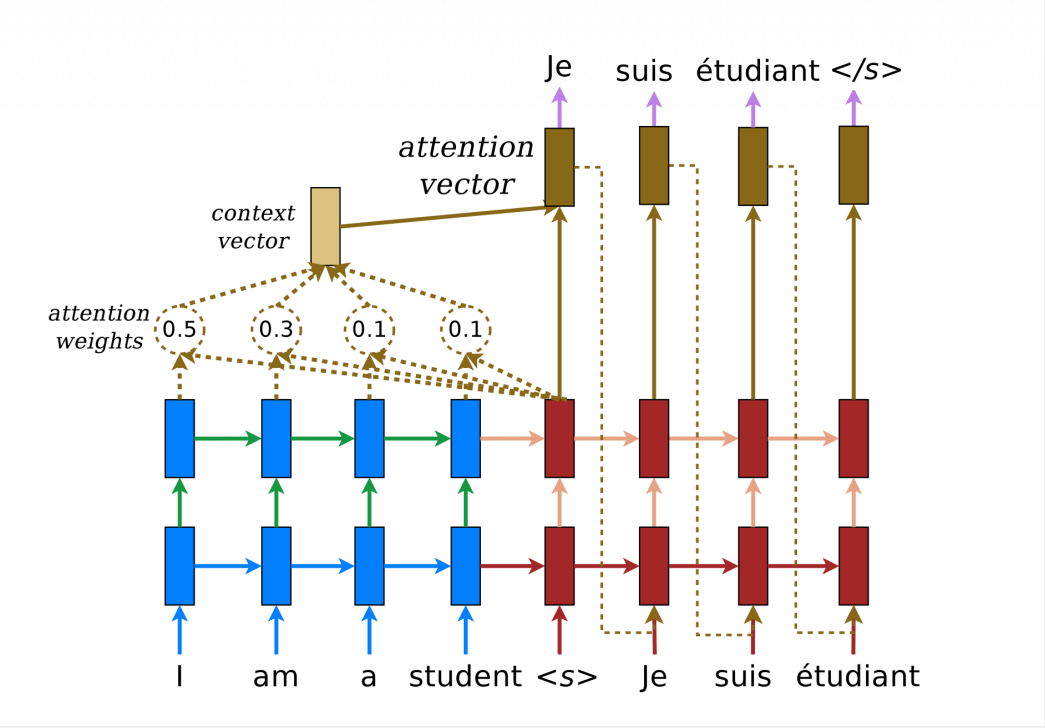
- a seq to seq model: $h_t^d = f_d(h_{t-1}^d, y_{t-1}, c)$
- we can set $c = h_t^e$, so the final state of the encode



M.-T. Luong. "Neural Machine Translation", PhD Thesis, Stanford CS Dept., 2016

Attention

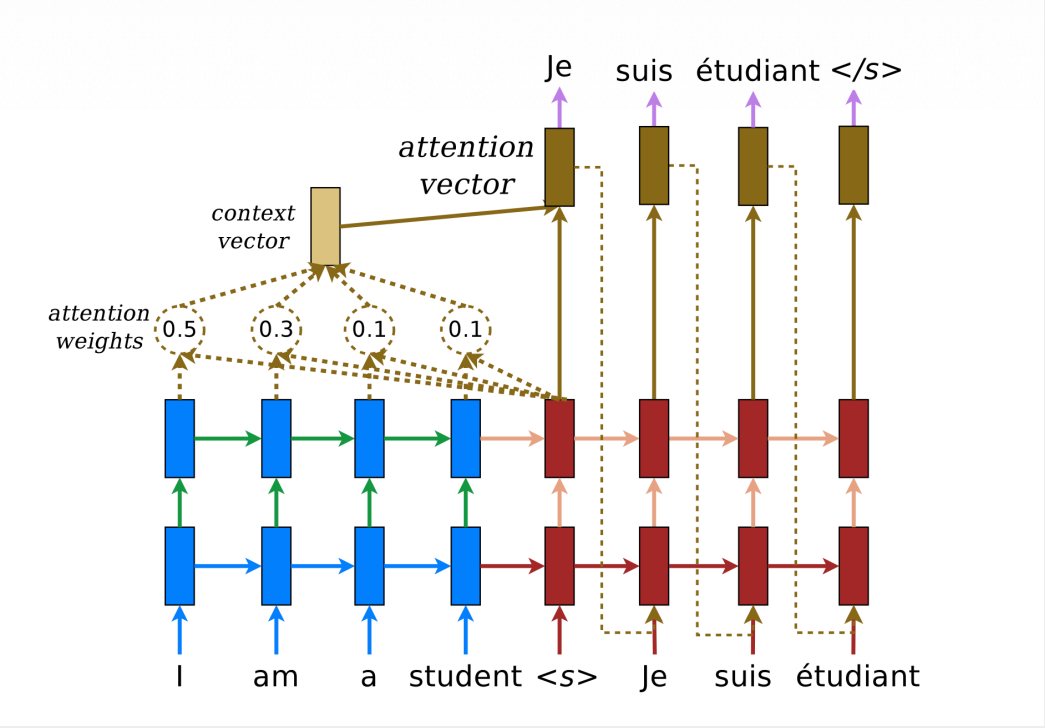
- a seq to seq model: $h_t^d = f_d(h_{t-1}^d, y_{t-1}, c)$
- we can set $c = h_t^e$, so the final state of the encode
- $\tilde{h}_t = \tanh(W_c[c_t; h_t])$



M.-T. Luong. "Neural Machine Translation", PhD Thesis, Stanford CS Dept., 2016

Attention

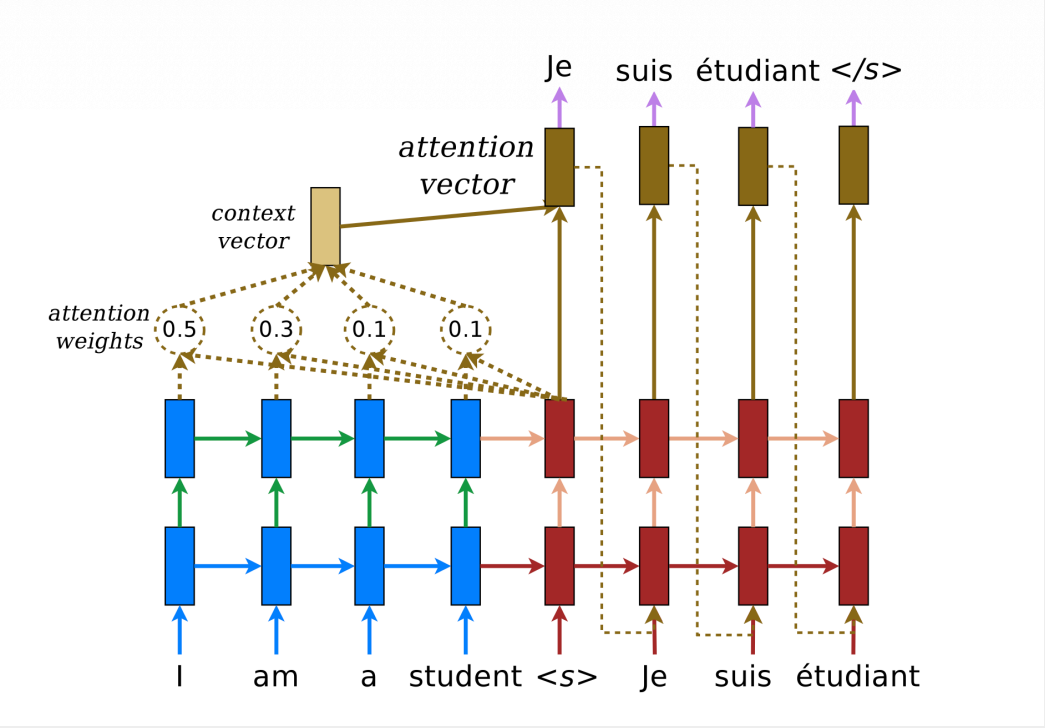
- a seq to seq model: $h_t^d = f_d(h_{t-1}^d, y_{t-1}, c)$
- we can set $c = h_t^e$, so the final state of the encode
- $\tilde{h}_t = \tanh(W_c[c_t; h_t])$
- this can result in poor performance, since the output does not have access to the input words themselves



M.-T. Luong. "Neural Machine Translation", PhD Thesis, Stanford CS Dept., 2016

Attention

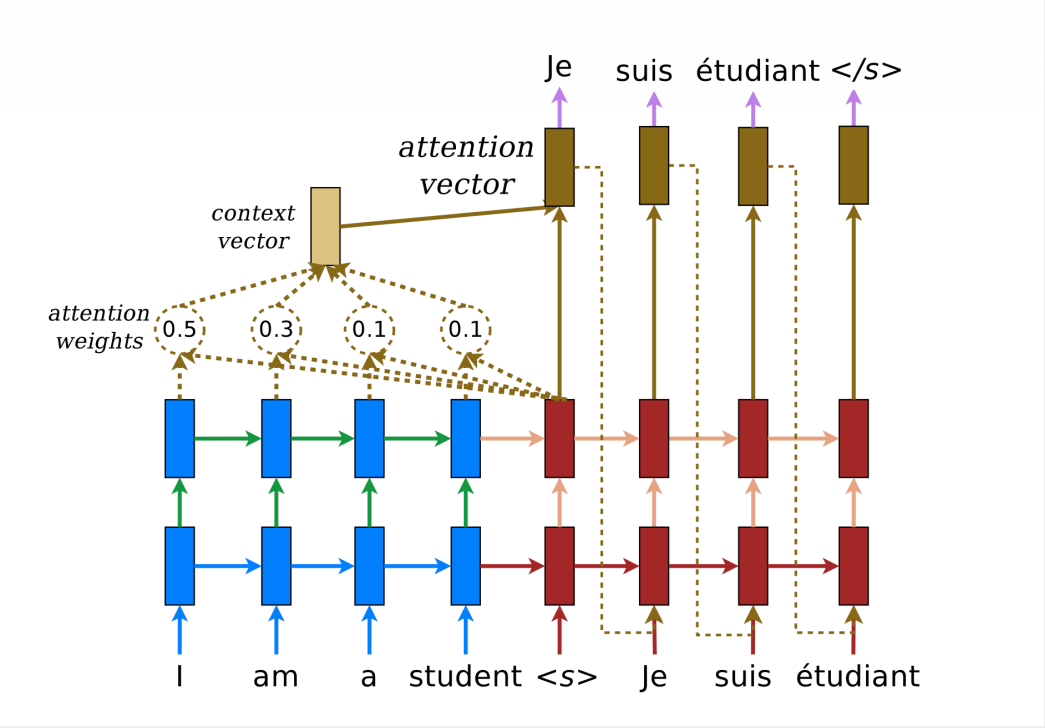
- a seq to seq model: $h_t^d = f_d(h_{t-1}^d, y_{t-1}, c)$
- we can set $c = h_t^e$, so the final state of the encode
- $\tilde{h}_t = \tanh(W_c[c_t; h_t])$
- this can result in poor performance, since the output does not have access to the input words themselves
- we are compressing alot of measing into a single vector



M.-T. Luong. "Neural Machine Translation", PhD Thesis, Stanford CS Dept., 2016

Attention

- a seq to seq model: $h_t^d = f_d(h_{t-1}^d, y_{t-1}, c)$
- we can set $c = h_t^e$, so the final state of the encode
- $\tilde{h}_t = \tanh(W_c[c_t; h_t])$
- this can result in poor performance, since the output does not have access to the input words themselves
- we are compressing alot of measing into a single vector
- we can avoid this bottleneck by allowing the output words to directly “look at” the input words

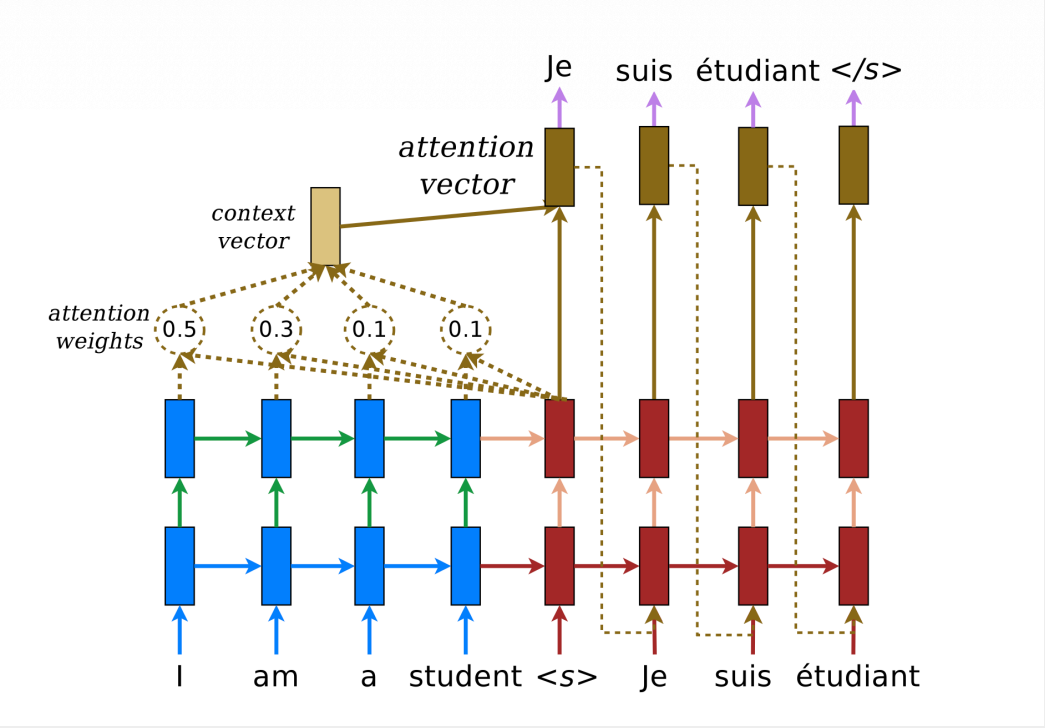


M.-T. Luong. "Neural Machine Translation", PhD Thesis, Stanford CS Dept., 2016

Attention

- a seq to seq model: $h_t^d = f_d(h_{t-1}^d, y_{t-1}, c)$
- we can set $c = h_t^e$, so the final state of the encode
- $\tilde{h}_t = \tanh(W_c[c_t; h_t])$
- this can result in poor performance, since the output does not have access to the input words themselves
- we are compressing alot of measing into a single vector
- we can avoid this bottleneck by allowing the output words to directly “look at” the input words
- use dynamic context vector:

$$c_t = \sum_{i=1}^{T} \alpha_i(h_{t-1}^d, h_{1:T}^e) h_i^e$$



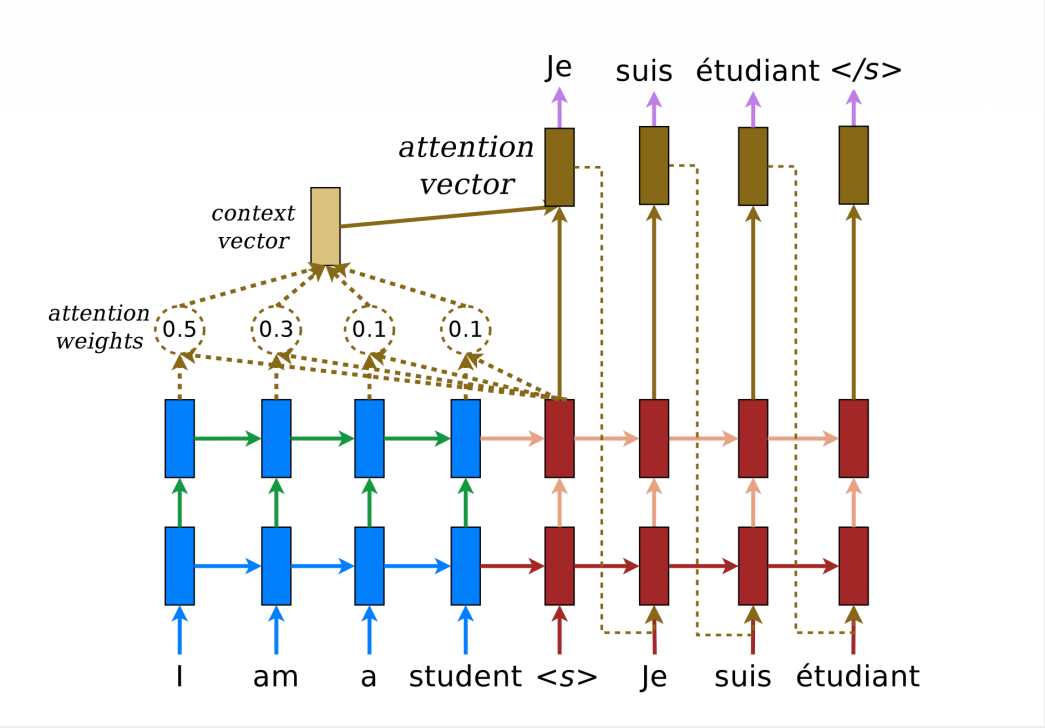
M.-T. Luong. "Neural Machine Translation", PhD Thesis, Stanford CS Dept., 2016

Attention

- a seq to seq model: $h_t^d = f_d(h_{t-1}^d, y_{t-1}, c)$
- we can set $c = h_t^e$, so the final state of the encode
- $\tilde{h}_t = \tanh(W_c[c_t; h_t])$
- this can result in poor performance, since the output does not have access to the input words themselves
- we are compressing alot of measing into a single vector
- we can avoid this bottleneck by allowing the output words to directly “look at” the input words
- use dynamic context vector:

$$c_t = \sum_{i=1}^{T} \alpha_i(h_{t-1}^d, h_{1:T}^e) h_i^e$$

- this is a weighted mean (soft dictionary look-up)



M.-T. Luong. "Neural Machine Translation", PhD Thesis, Stanford CS Dept., 2016

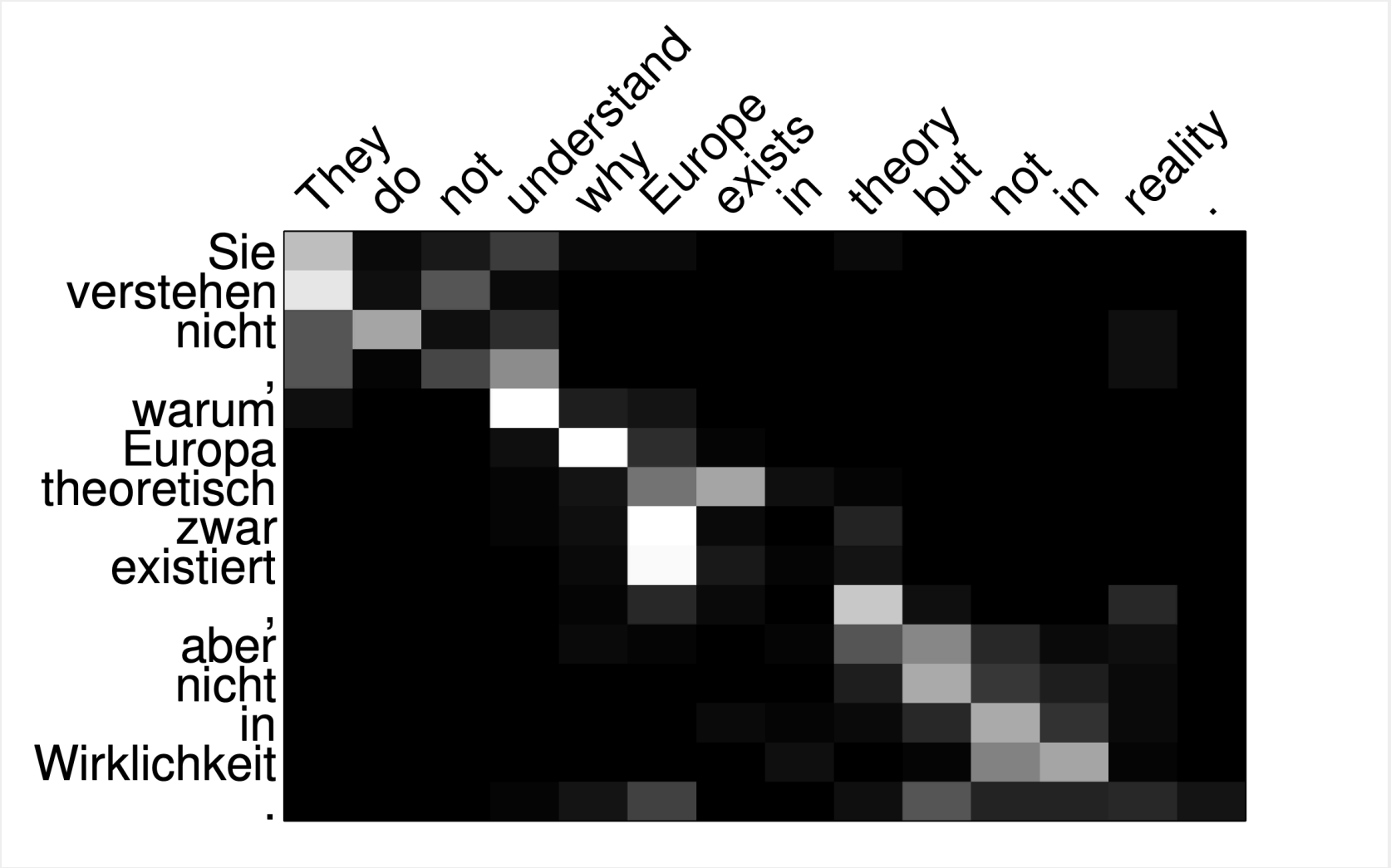
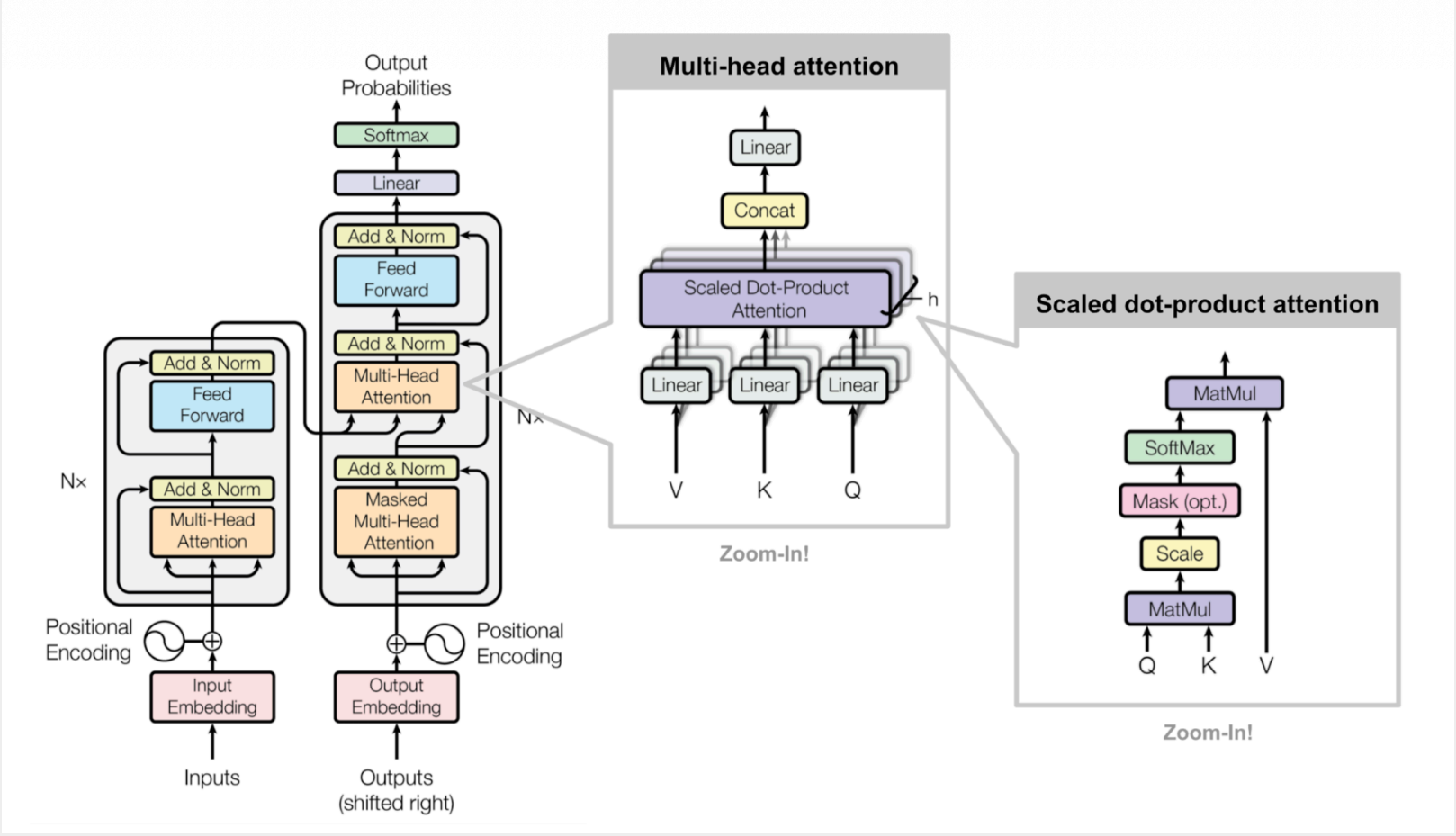
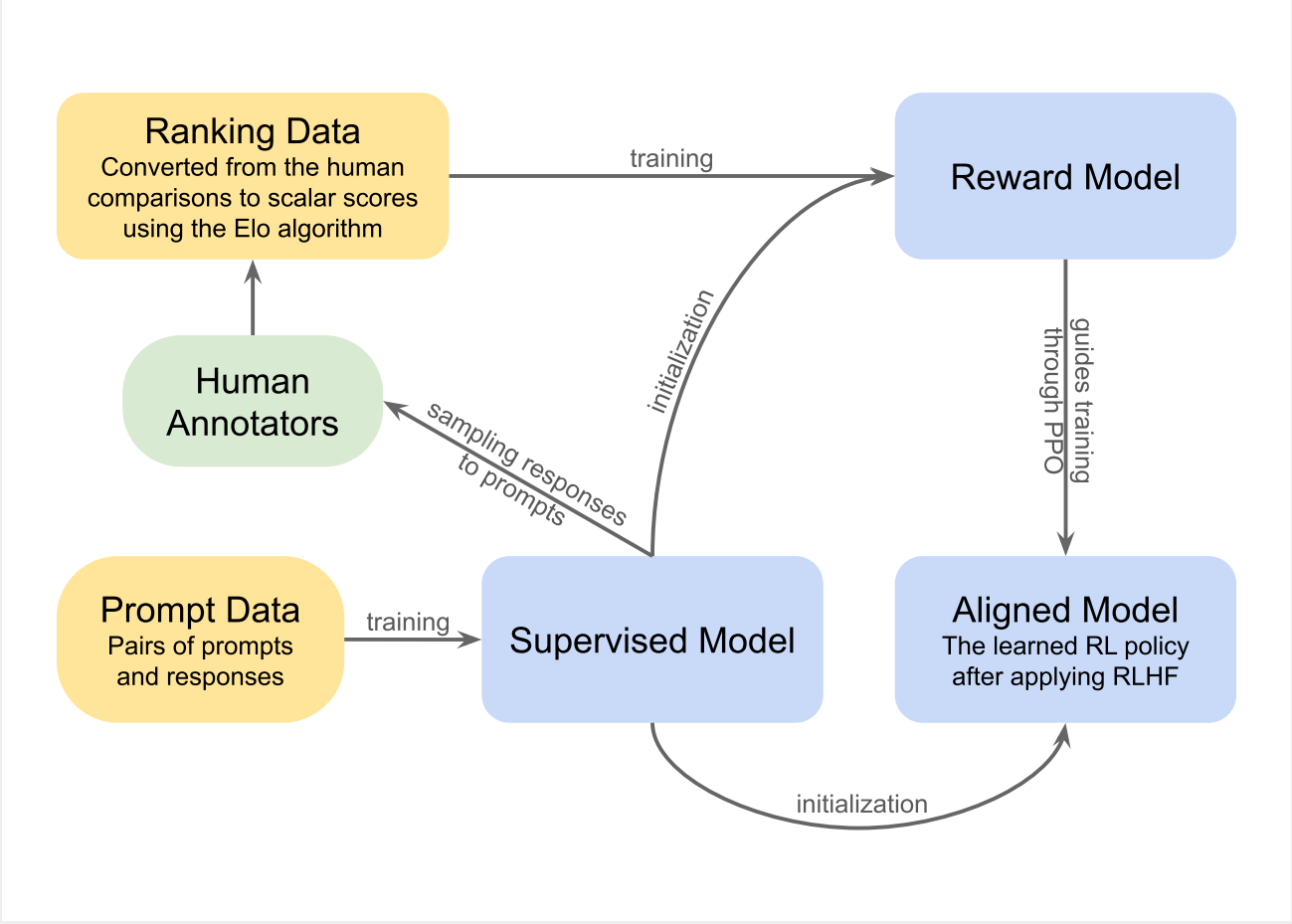


Illustration of the attention heatmaps generated while translating a sentence from German to English. From M.-T. Luong. "Neural Machine Translation", PhD Thesis, Stanford CS Dept., 2016

Transformer



Reinforcement Learning from Human Feedback (RLHF)



Note: when do you use RL? if you don't know the loss function yourself (works with thumbs-up/thumbs-down)

Reasoning Models: CoT

Regular prompting

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: The answer (arabic numerals) is

(Output) 8 ✗

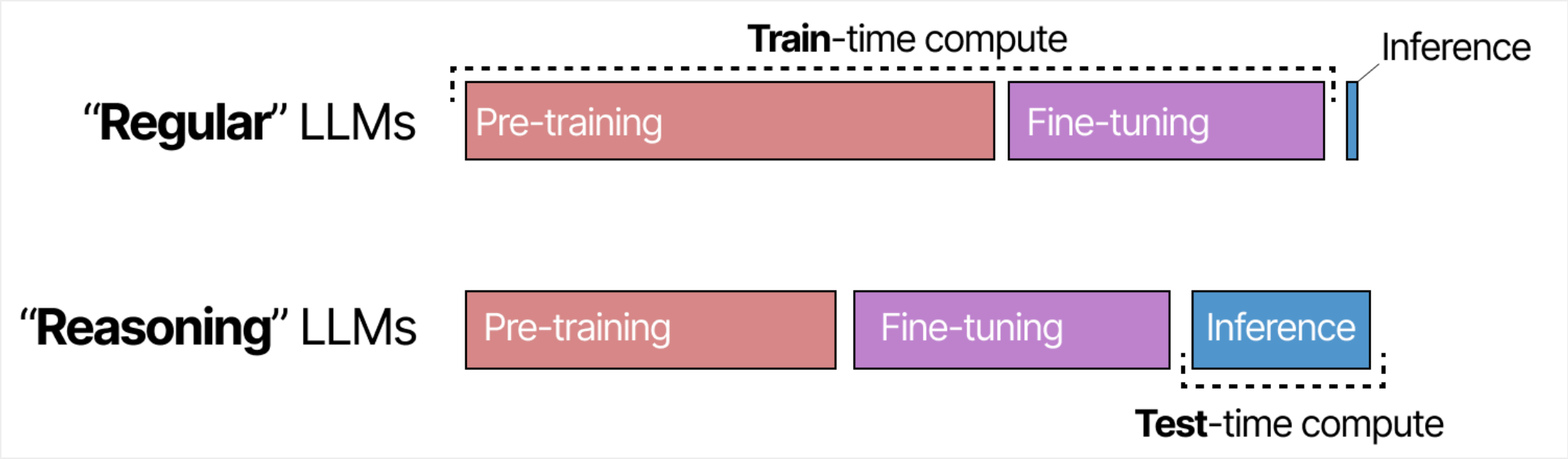
Chain-of-thought prompting

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: ***Let's think step by step.***

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

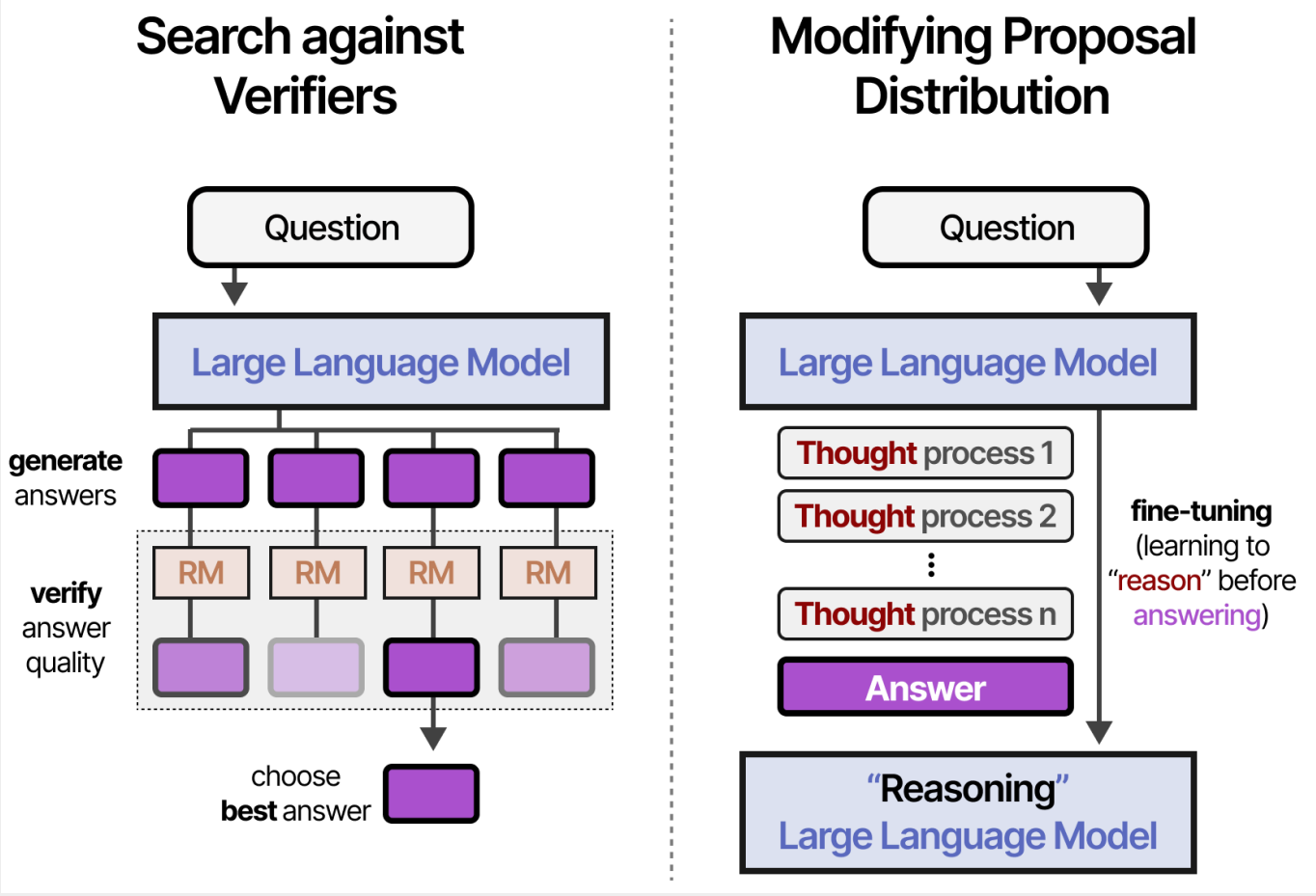
An example of classic CoT prompting from the 2022 Large Language Models are Zero-Shot Reasoners paper (https://arxiv.org/abs/2205.11916).

Reasoning Models



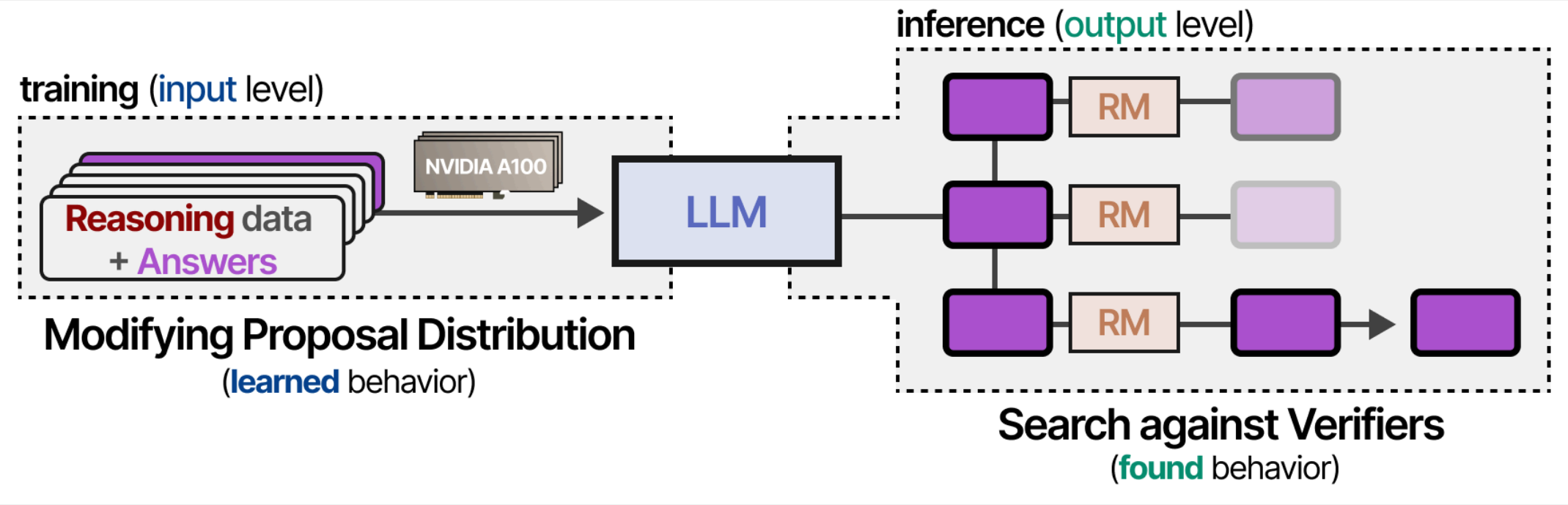
Source: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-reasoning-llms

Reasoning Models



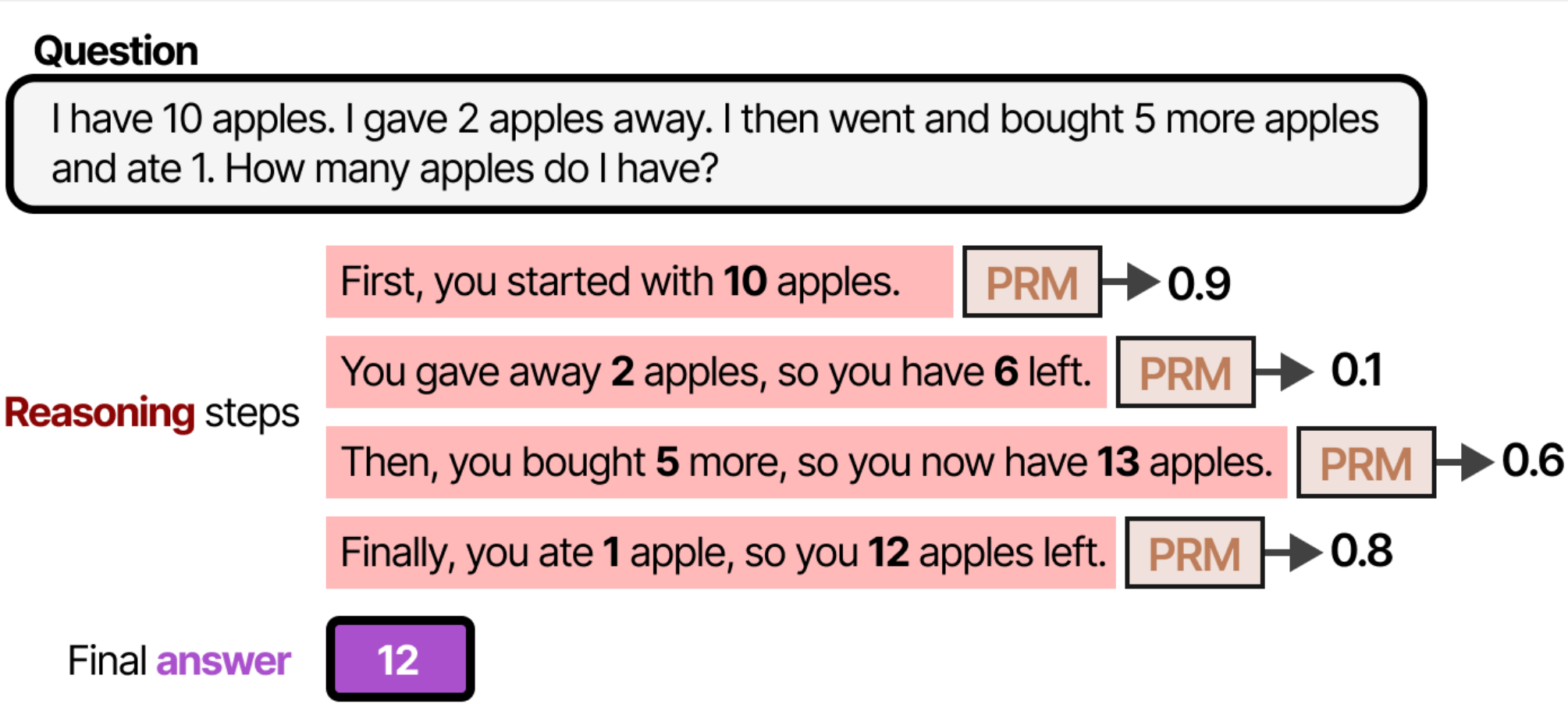
Source: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-reasoning-llms

Reasoning Models



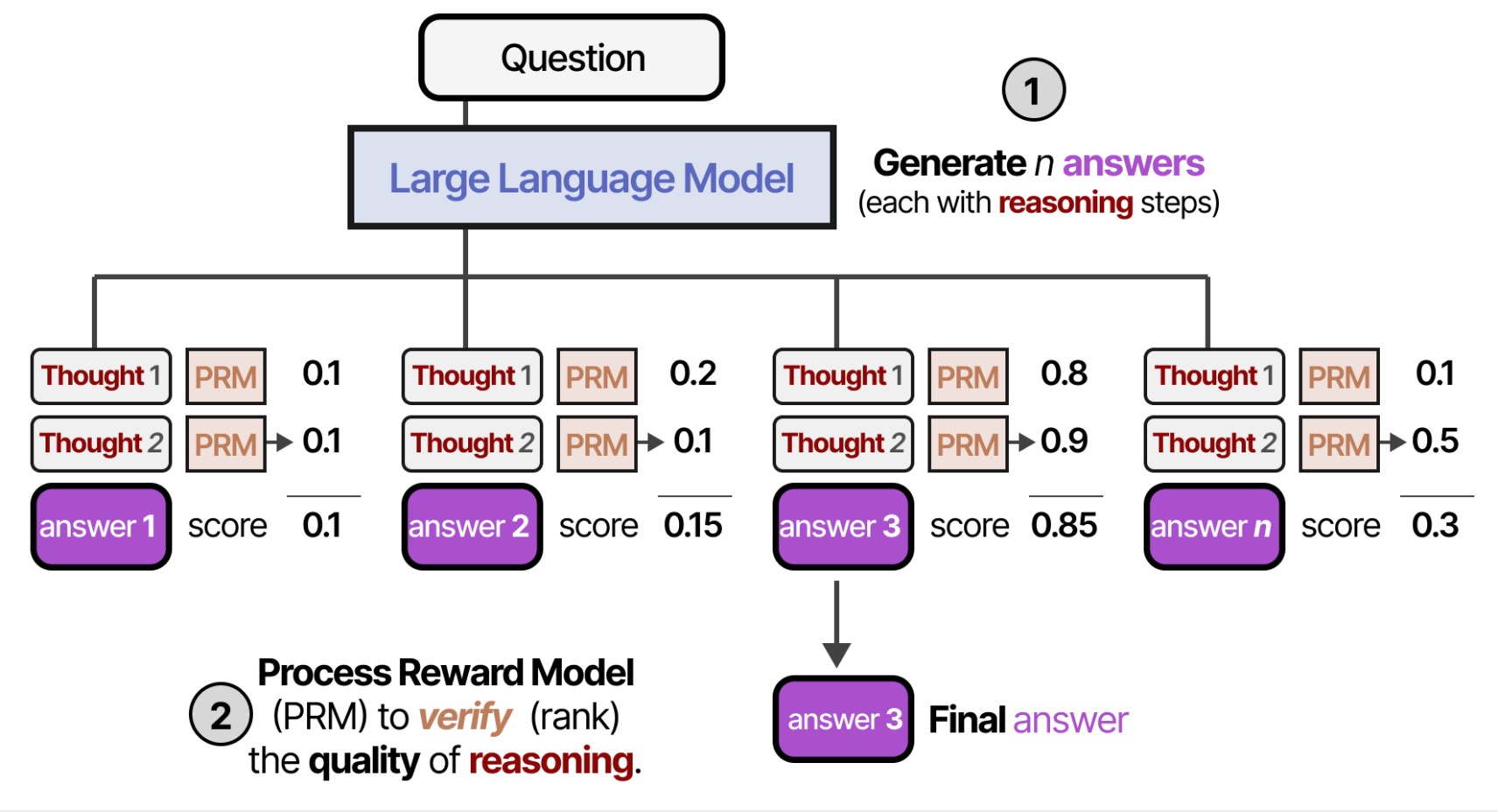
Source: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-reasoning-llms

Reasoning Models



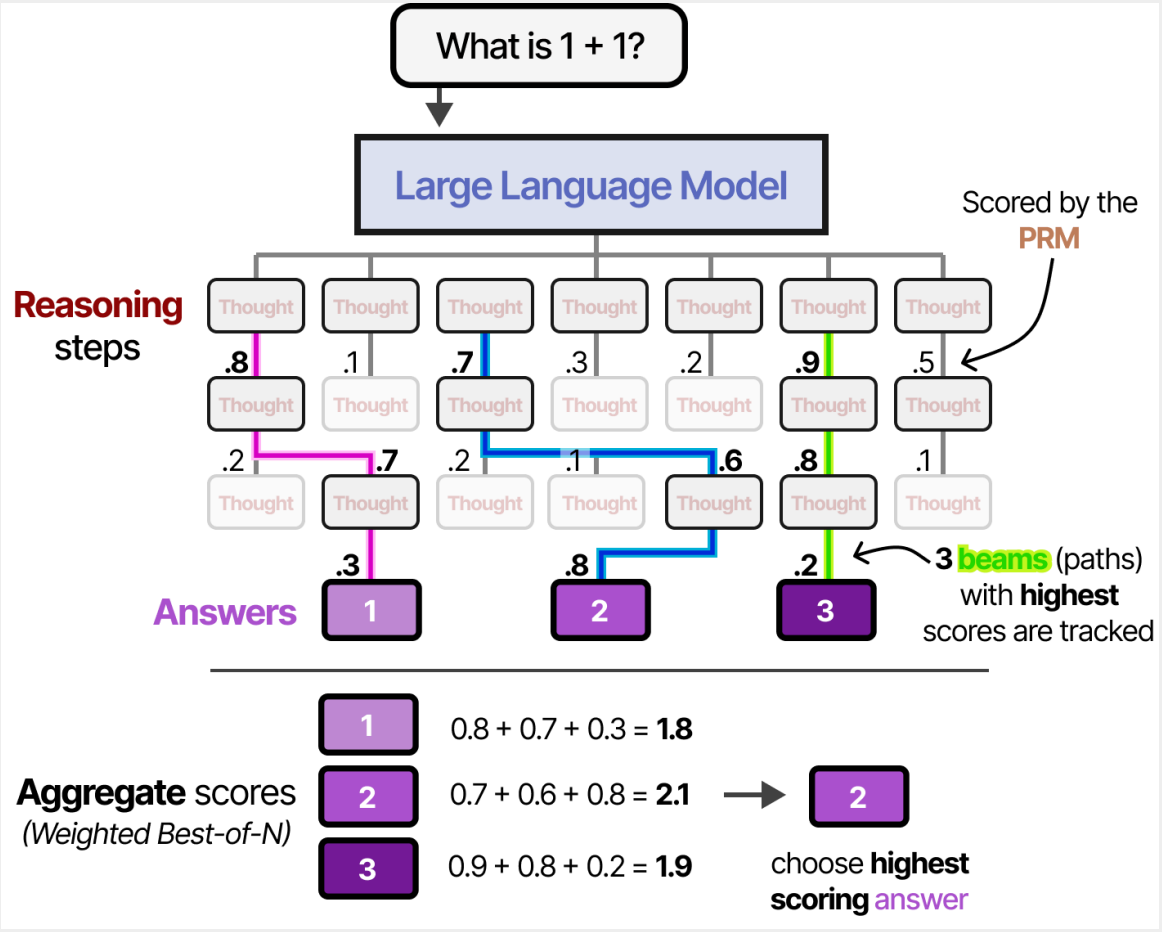
Source: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-reasoning-llms

Reasoning Models



Source: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-reasoning-llms

Reasoning Models: beam search with PRM



Source: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-reasoning-llms

Reasoning Models: 4 ways to improve reasoning

1. inference-time scaling: increasing computational resources during inference
   - Chain of thought prompting is one version of that
   - Another one is best of N: Generating multiple solutions and using a process-based verifier reward model (it has to be separately trained) to select the best response. Such a model learns how effective reasoning should look like

Reasoning Models: 4 ways to improve reasoning

1. inference-time scaling: increasing computational resources during inference
   - Chain of thought prompting is one version of that
   - Another one is best of N: Generating multiple solutions and using a process-based verifier reward model (it has to be separately trained) to select the best response. Such a model learns how effective reasoning should look like
2. Pure reinforcement learning (RL)
   - DeepSeek-R1-Zero was trained exclusively with RL and discovered that reasoning emerges as a behavior from that
     - The accuracy reward uses the LeetCode compiler to verify coding answers and a deterministic system to evaluate mathematical responses.
     - The format reward relies on an LLM judge to ensure responses follow the expected format, such as placing reasoning steps inside tags.

Reasoning Models: 4 ways to improve reasoning

3. Supervised finetuning and reinforcement learning (SFT + RL)
   - multiple rounds of instruction based fine-tuning (similar to RLHF) and RL with accuracy rewards
   - datasets of CoT data are generated and used for fine-tuning

Reasoning Models: 4 ways to improve reasoning

3. Supervised finetuning and reinforcement learning (SFT + RL)
   - multiple rounds of instruction based fine-tuning (similar to RLHF) and RL with accuracy rewards
   - datasets of CoT data are generated and used for fine-tuning
4. Pure supervised finetuning (SFT) and distillation
   - train smaller models on output of larger models

Mixture of Experts (MoE)

- in regular LLMs (e.g. GPT-3), every parameter is used to process every single token

Mixture of Experts (MoE)

- in regular LLMs (e.g. GPT-3), every parameter is used to process every single token
- MoE: break up network into sub-networks (“experts”) and train a gating network

Mixture of Experts (MoE)

- in regular LLMs (e.g. GPT-3), every parameter is used to process every single token
- MoE: break up network into sub-networks (“experts”) and train a gating network
- the gating network learns to route tokens to different networks

Mixture of Experts (MoE)

- in regular LLMs (e.g. GPT-3), every parameter is used to process every single token
- MoE: break up network into sub-networks (“experts”) and train a gating network
- the gating network learns to route tokens to different networks
- mathematically this is a weighted combination, but the actual implementations select Top-K, leading

to just K sub-networks being used for the processing

Mixture of Experts (MoE)

- in regular LLMs (e.g. GPT-3), every parameter is used to process every single token
- MoE: break up network into sub-networks (“experts”) and train a gating network
- the gating network learns to route tokens to different networks
- mathematically this is a weighted combination, but the actual implementations select Top-K, leading

to just K sub-networks being used for the processing

- allows to train much larger models, since at any time, only a smaller part gets activated

Building Chatbots

Request to OpenAI **Completions** API:

```
curl https://api.openai.com/v1/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY"
\
  -d '{
    "model": "gpt-3.5-turbo-instruct",
    "prompt": "Say this is a test",
    "max_tokens": 7,
    "temperature": 0
  }'
```

And you'd get back this:

```
{
  "id": "cmpl-uqkvlQyYK7bGYrRHQ0eXlWi7",
  "object": "text_completion",
  "created": 1589478378,
  "model": "gpt-3.5-turbo-instruct",
  "system_fingerprint": "fp_44709d6fcb",
  "choices": [
    {
      "text": "\n\nThis is indeed a test",
      "index": 0,
      "logprobs": null,
      "finish_reason": "length"
    }
  ],
  "usage": {
    "prompt_tokens": 5,
    "completion_tokens": 7,
    "total_tokens": 12
  }
}
```

Building Chatbots

For the **Chat Completions** API, we send/recieve a list of messages:

```
curl https://api.openai.com/v1/chat/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -d '{
    "model": "gpt-5.2",
    "messages": [
      {
        "role": "developer",
        "content": "You are a helpful assistant."
      },
      {
        "role": "user",
        "content": "Hello!"
      }
    ]
  }'
```

Building Chatbots

- we can call the OpenAI API to get a completion
- how to we have a conversation?

```
model = ChatOpenAI()
messages = []
while True:
  user_input = input("User: ")
  messages.append(("user", user_input))
  response = model.invoke(messages)
  messages.append(("assistant", response.content))
  print(f"AI: {response.content}")
```

Building Chatbots: LangGraph

```python
from langchain_openai import ChatOpenAI
from langgraph.checkpoint.memory import MemorySaver
from langgraph.graph import START, MessagesState, StateGraph
model = ChatOpenAI()
graph = (
  StateGraph(MessagesState)
  .add_node(
    "chatbot",
    lambda state: {"messages": model.invoke(state["messages"])}
  )
  .add_edge(START, "chatbot")
  .compile(checkpointer=MemorySaver())
)
config = {"configurable": {"thread_id": "1"}}
while True:
  user_input = input("User: ")
  response = graph.invoke({"messages": [("user", user_input)]}, config)
  print(f"AI: {response['messages'][-1].content}")
```

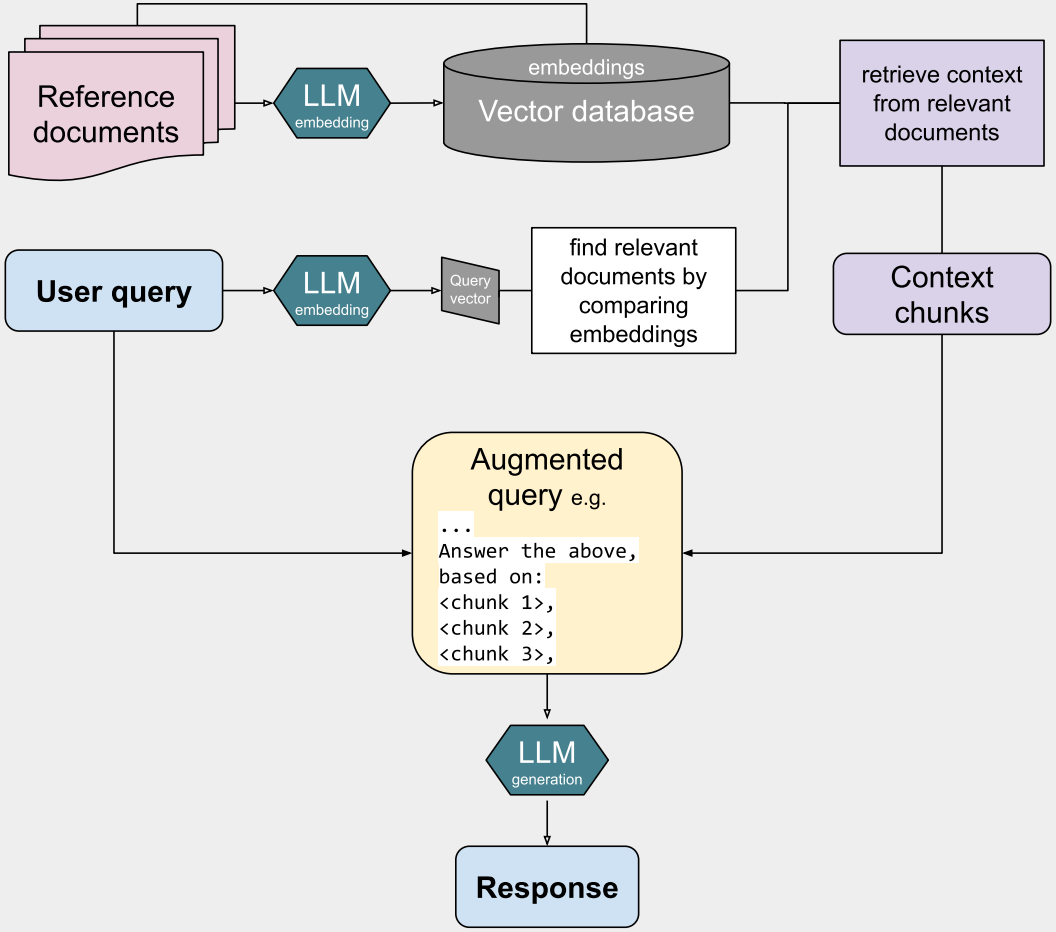
Building Chatbots: LangGraph (functional)

```python
from langgraph.checkpoint.memory import MemorySaver
from langgraph.func import entrypoint

@entrypoint(checkpointer=MemorySaver())
def chat_workflow(new_message: str, previous: list = None):
    history = previous or []
    history.append(("user", new_message))
    response = model.invoke(history)
    history.append(("assistant", response.content))
    return history

config = {"configurable": {"thread_id": "1"}}
while True:
    user_input = input("User: ")
    output = chat_workflow.invoke(user_input, config=config)
    print("AI: ", output[-1][1])
```

Exercise 1

- see `exercises/01_chatbot.ipynb`

Retrieval Augemented Generation



Source: https://en.wikipedia.org/wiki/Retrieval-augmented_generation#/media/File:RAG_diagram.svg (retrieved on 29.12.2025)

Exercise 2

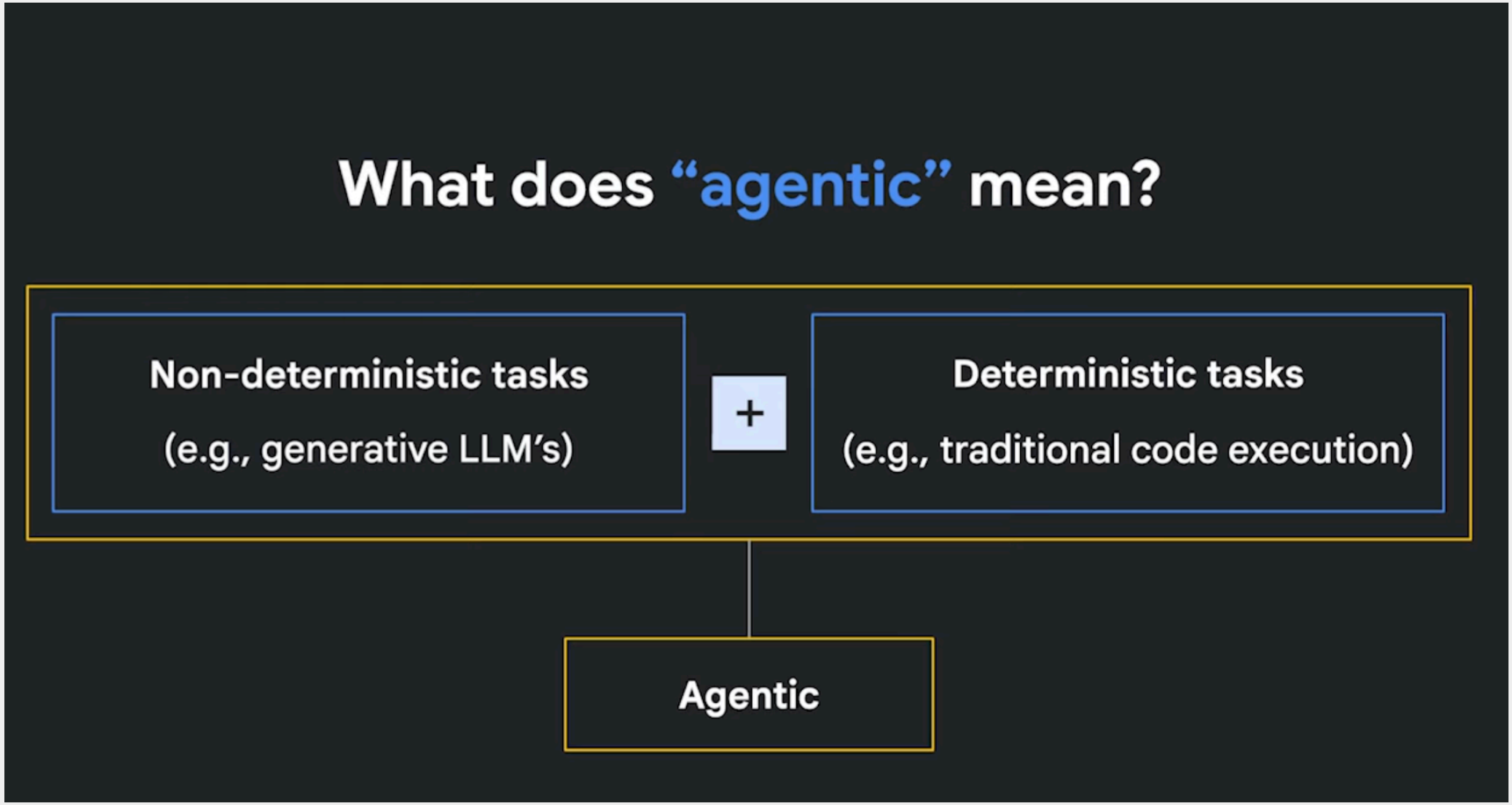
- see `exercises/02_rag.ipynb`

Agenda

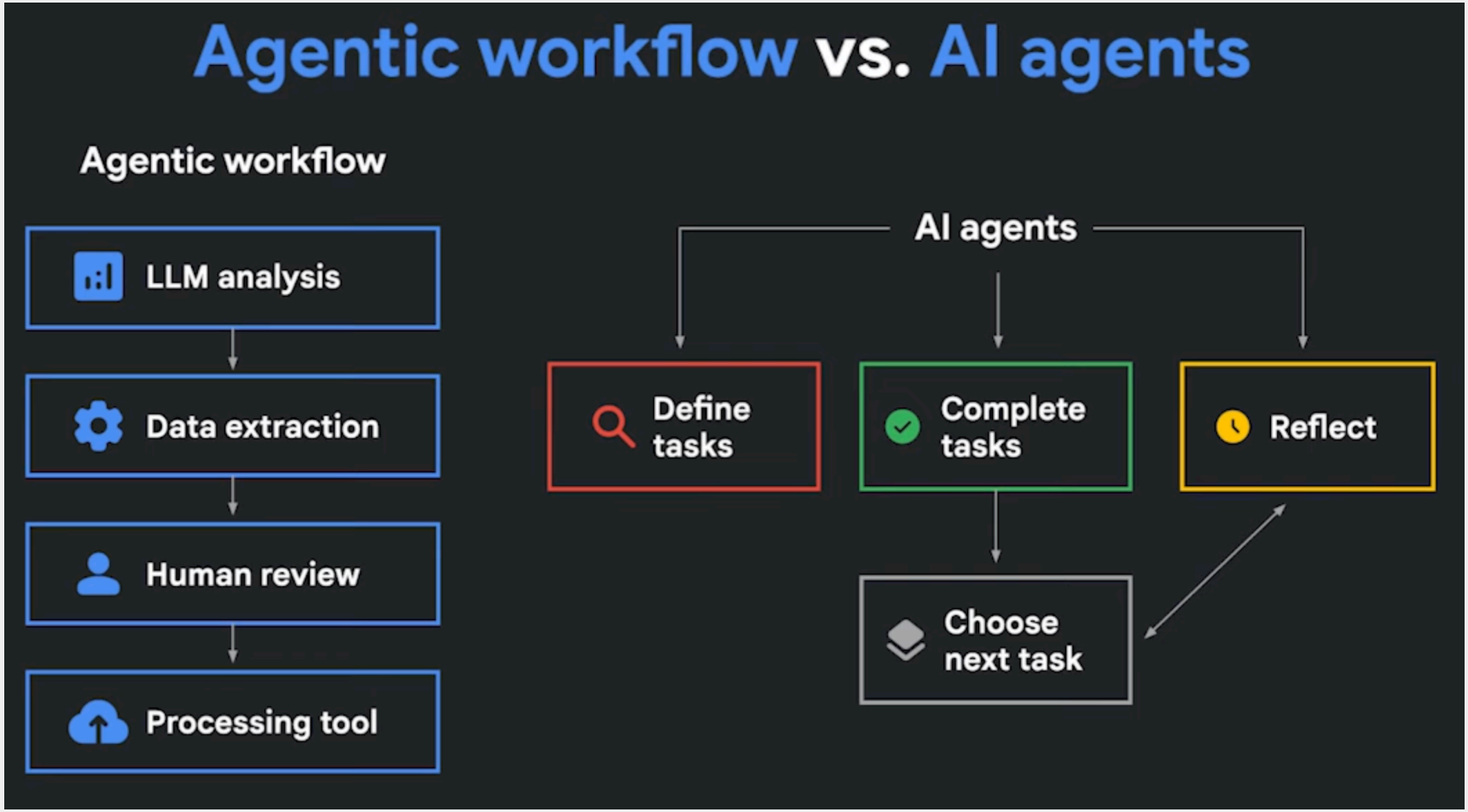
1	Introduction	09:00
2	LLM basics	09.15
3	**Agents**	10.00
4	Lab 1: Deep research	11.00
5	Lunch	12.00
6	MCP	13.00
7	Agents: Advanced Concepts	14.00
8	Lab 2: MCP + advanced agents	15.15
9	Conclusion	16.45

What does agentic mean?

What does agentic mean?



Workflows vs. agents



Tool calling

- tool calling unlocks interaction with the external world

The New England Journal of Medicine is a registered trademark of [QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society] the MMS.

Out of 1400 participants, 400 (or [Calculator(400 / 1400) → 0.29] 29%) passed the test.

The name derives from "la tortuga", the Spanish word for [MT("tortuga") → turtle] turtle.

The Brown Act is California's law [WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.] that requires legislative bodies, like city councils, to hold their meetings open to the public.

Figure 1: Exemplary predictions of Toolformer. The model autonomously decides to call different APIs (from top to bottom: a question answering system, a calculator, a machine translation system, and a Wikipedia search engine) to obtain information that is useful for completing a piece of text.

From 'Toolformer' paper (2022)

Tool calling

- tool calling unlocks interaction with the external world
- how does it work?
  - fine-tune to produce certain structured outputs
  - json strings that represent tool calls

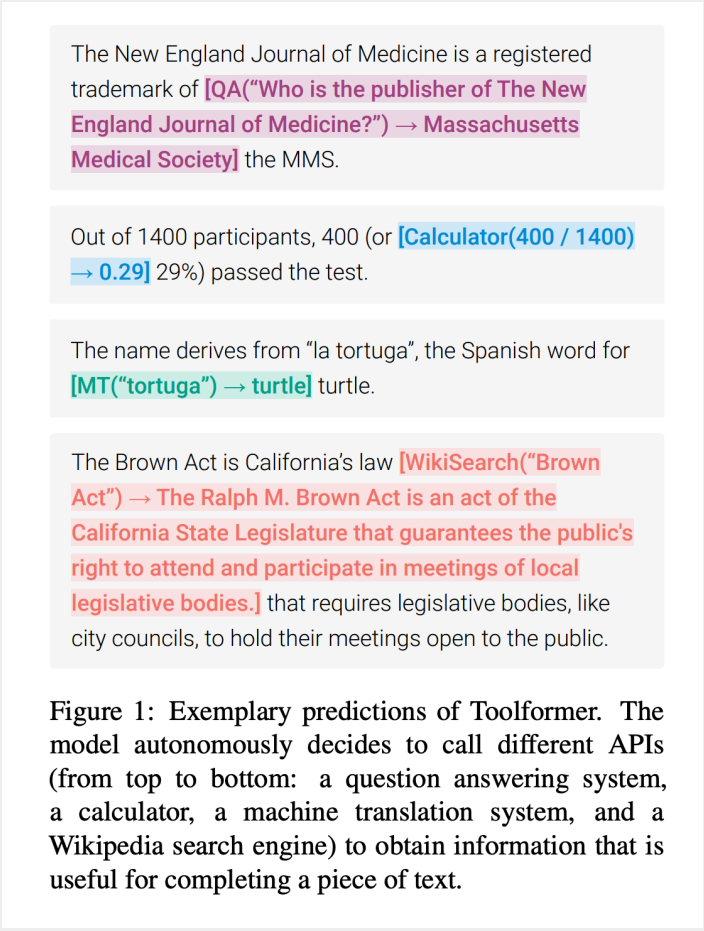


Figure 1: Exemplary predictions of Toolformer. The model autonomously decides to call different APIs (from top to bottom: a question answering system, a calculator, a machine translation system, and a Wikipedia search engine) to obtain information that is useful for completing a piece of text.

From 'Toolformer' paper (2022)

Tool calling

```
curl https://api.openai.com/v1/chat/completions \
-H "Content-Type: application/json" \
-H "Authorization: Bearer $OPENAI_API_KEY" \
-d '{
  "model": "gpt-4.1",
  "messages": [
    {
      "role": "user",
      "content": "What is the weather like in Boston today?"
    }
  ],
  "tools": [
    {
      "type": "function",
      "function": {
        "name": "get_current_weather",
        "description": "Get the current weather in a given location",
        "parameters": {
          "type": "object",
          "properties": {
            "location": {
              "type": "string",
              "description": "The city and state, e.g. San Francisco, CA"
            },
            "unit": {
              "type": "string", "enum": ["celsius", "fahrenheit"]
            }
          },
          "required": ["location"]
  ...
}'
```

Tool calling

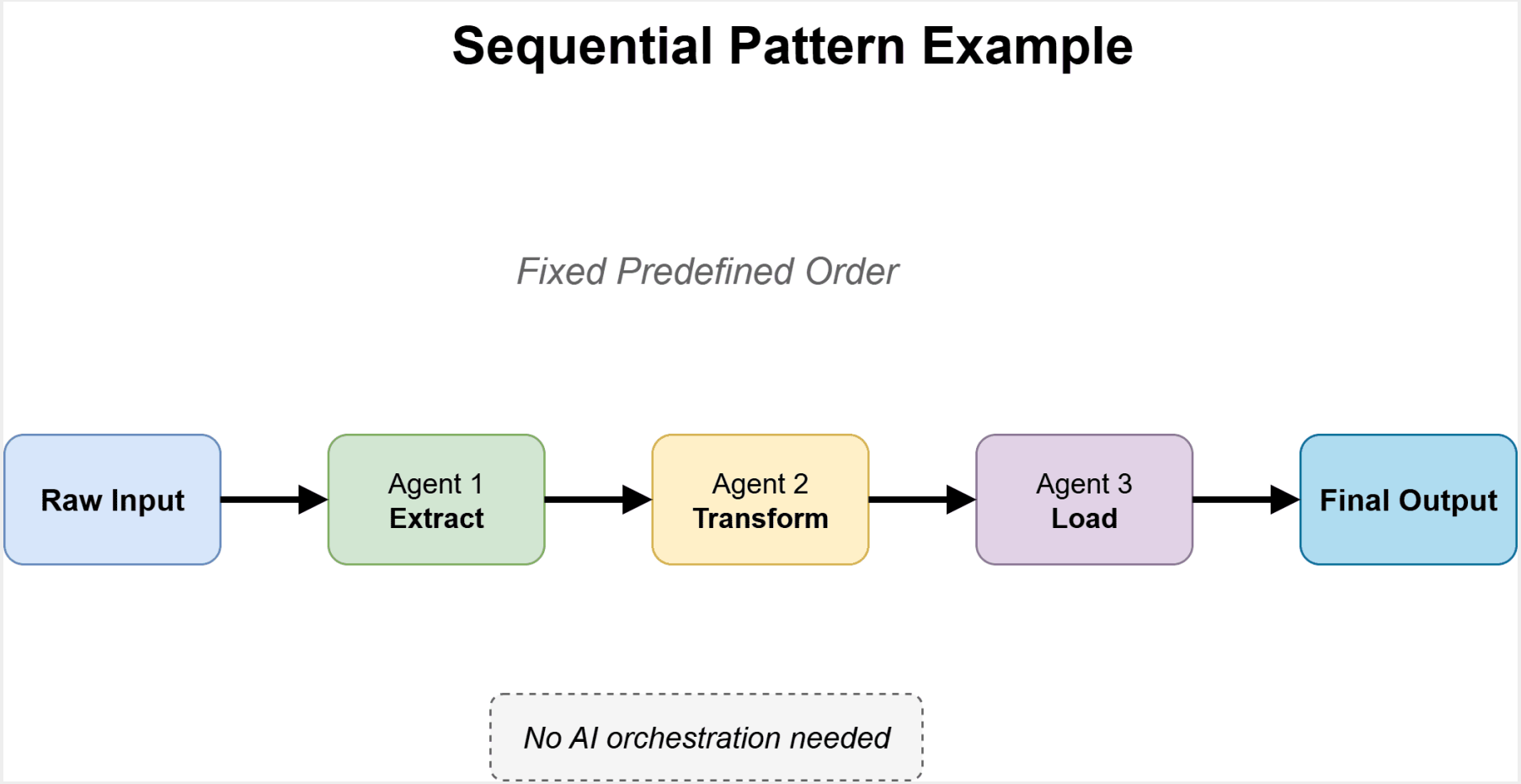

```
{
  "id": "chatcmpl-abc123",
  "object": "chat.completion",
  "created": 1699896916,
  "model": "gpt-4o-mini",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": null,
        "tool_calls": [
          {
            "id": "call_abc123",
            "type": "function",
            "function": {
              "name": "get_current_weather",
              "arguments": "{\n\"location\":
\"Boston, MA\"\n}"
            }
          }
        ]
      },
      "logprobs": null,
      "finish_reason": "tool_calls"
    }
  ],
  "usage": {
    "prompt_tokens": 82,
    "completion_tokens": 17,
    "total_tokens": 99,
    "completion_tokens_details": {
      "reasoning_tokens": 0,
      "accepted_prediction_tokens": 0,
      "rejected_prediction_tokens": 0
    }
  }
}
```

Exercise 3

- see `exercises/03_tool_calling.ipynb`
  - write from scratch
  - using langgraph

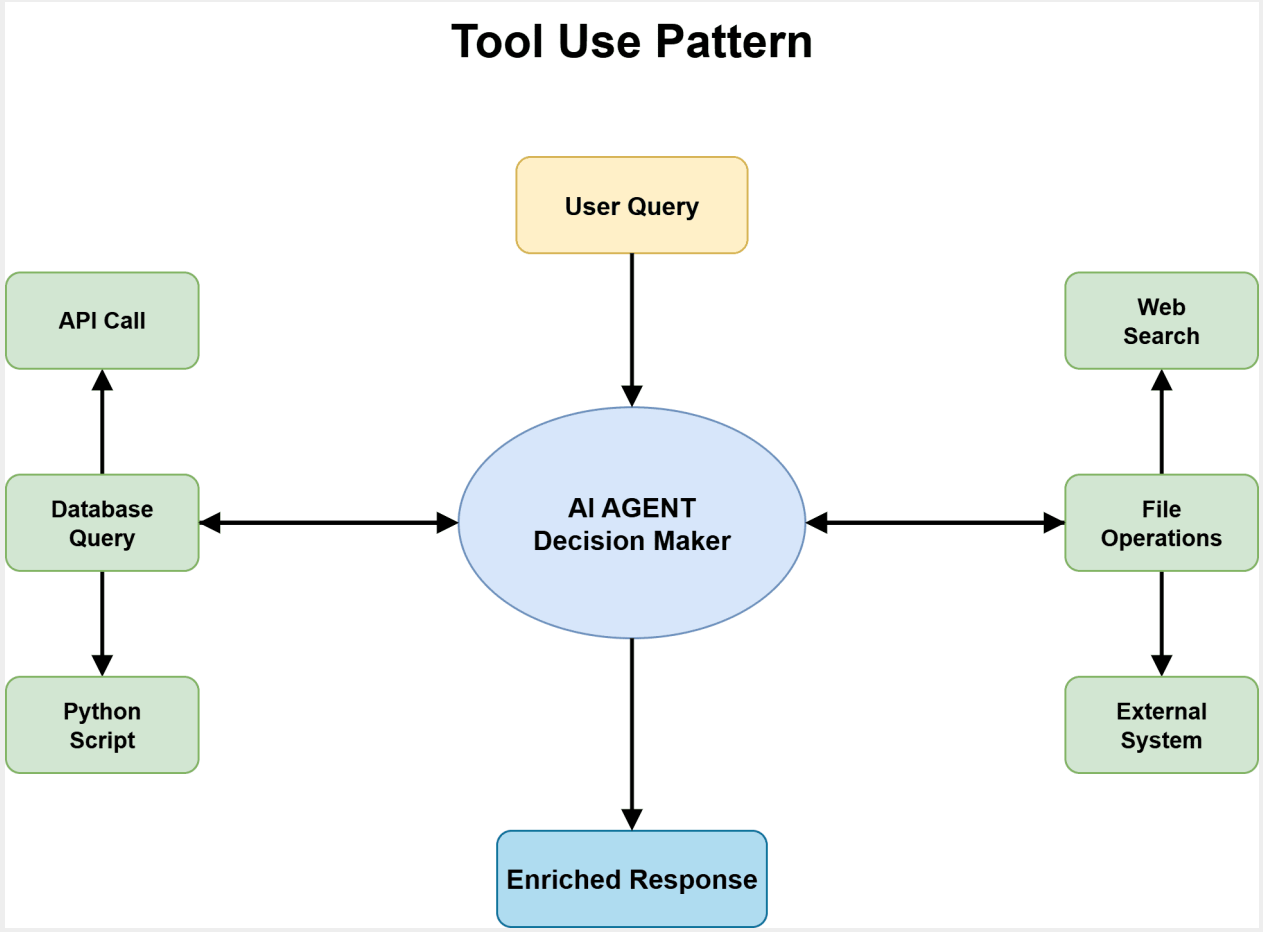

Coffee break

Agentic Design Patterns: Sequential workflow



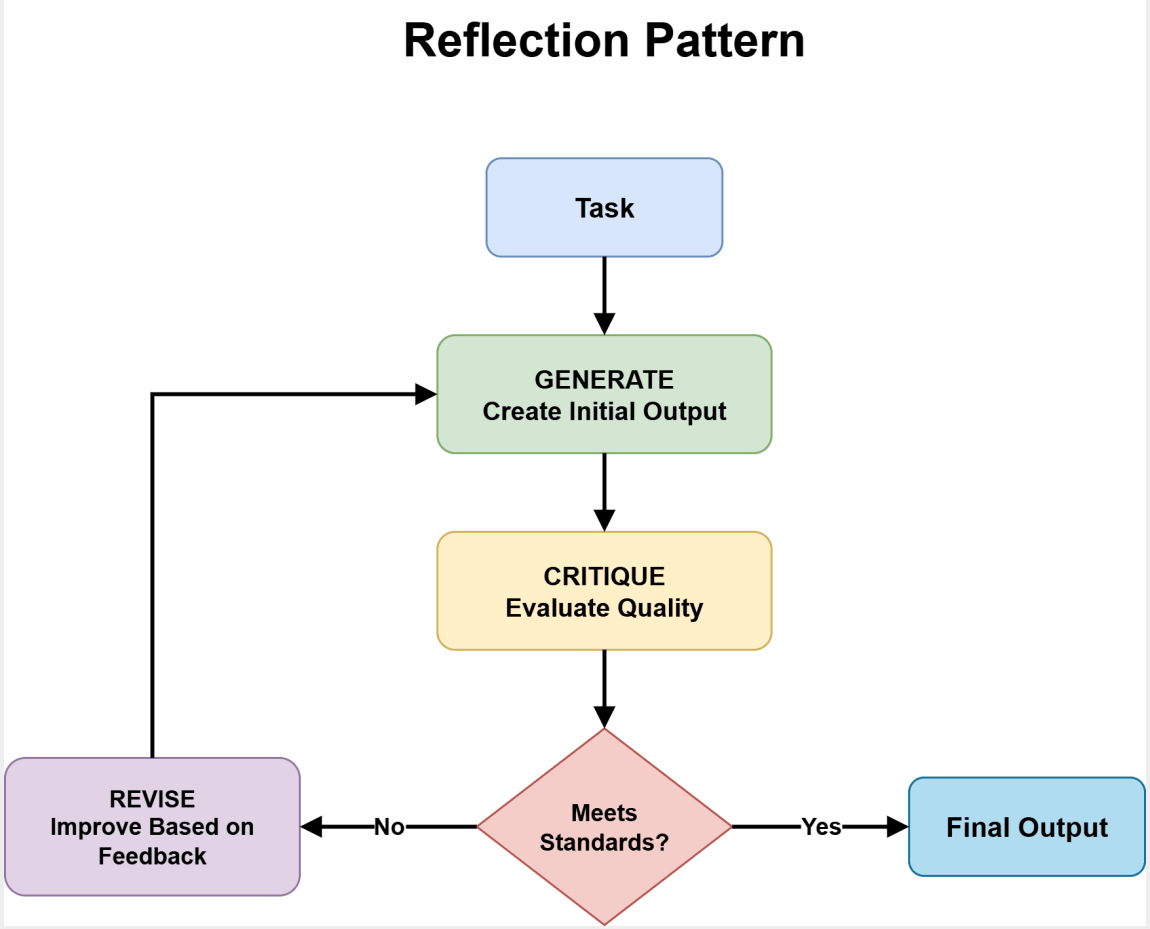
Source: https://machinelearningmastery.com/7-must-know-agentic-ai-design-patterns

Agentic Design Patterns: Tool use



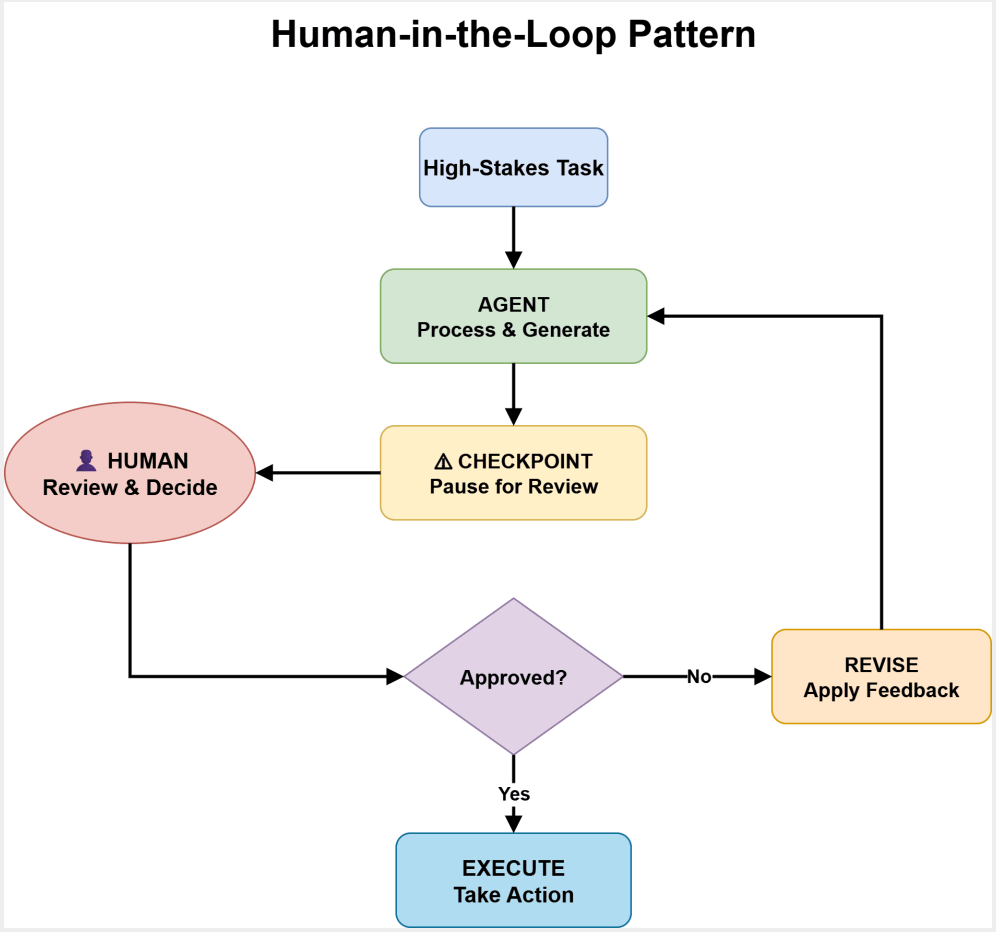
Source: https://machinelearningmastery.com/7-must-know-agentic-ai-design-patterns

Agentic Design Patterns: Reflection



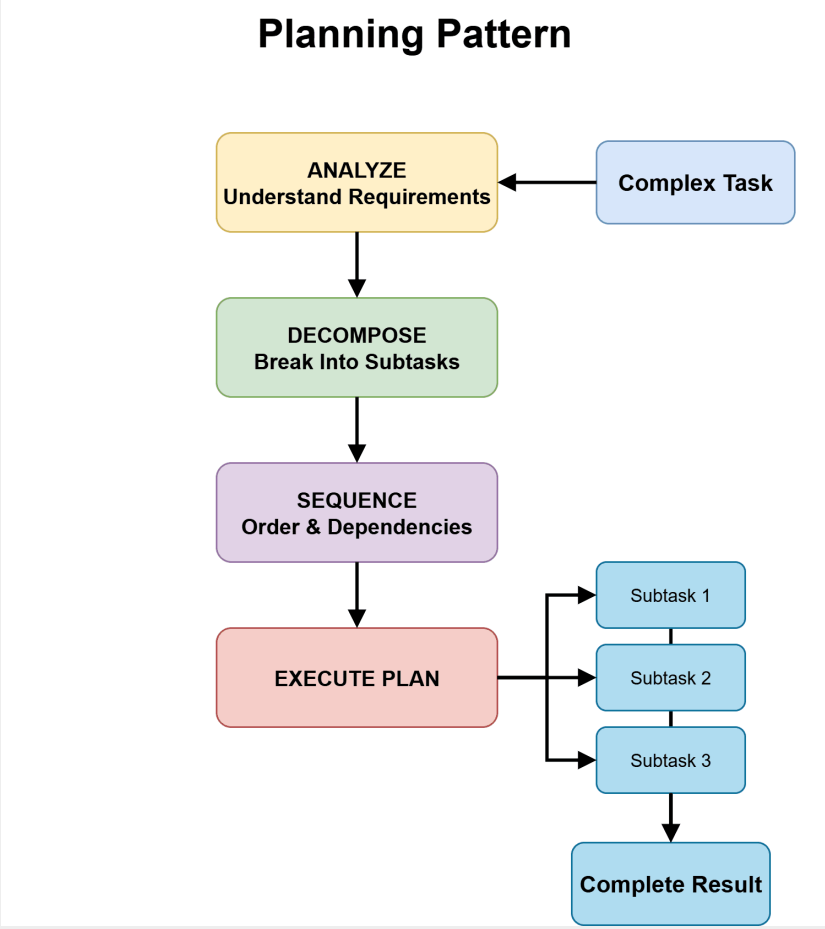
Source: https://machinelearningmastery.com/7-must-know-agentic-ai-design-patterns

Agentic Design Patterns: HITL pattern



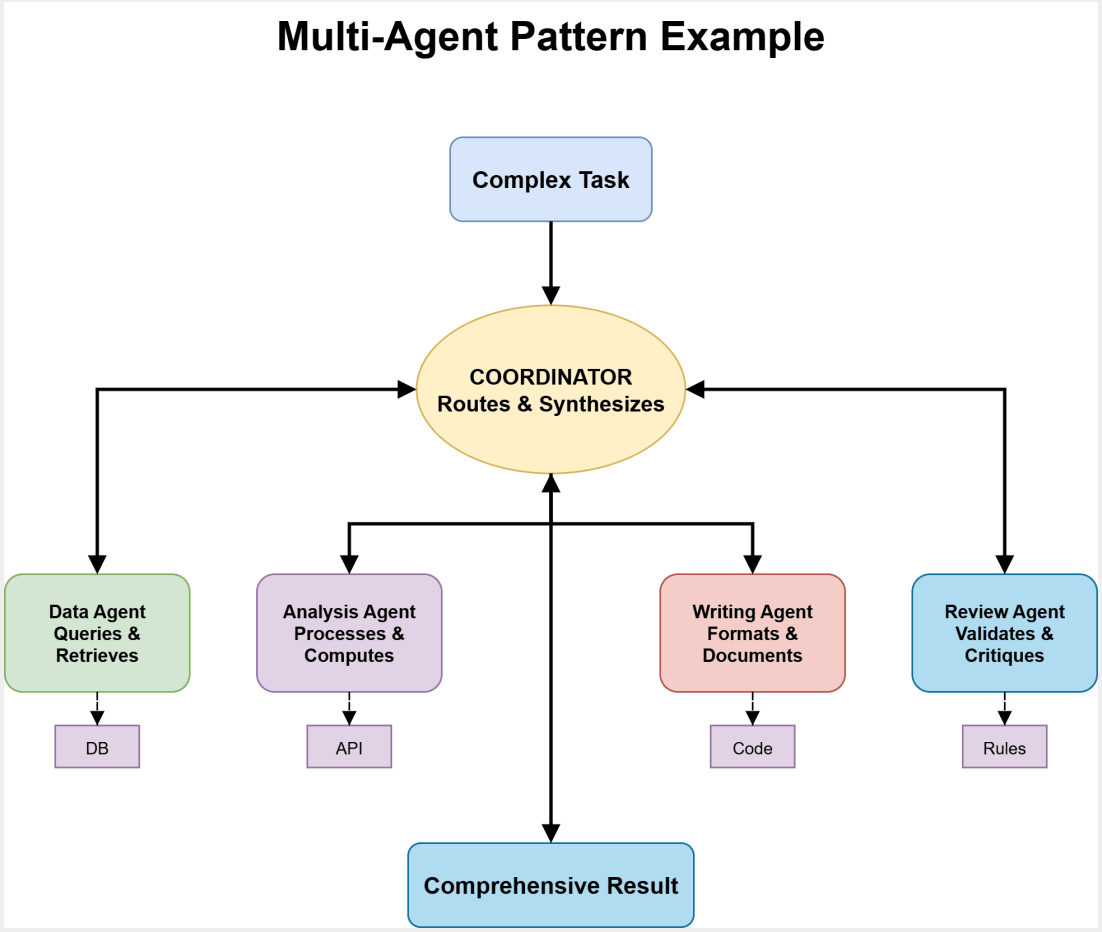
Source: https://machinelearningmastery.com/7-must-know-agentic-ai-design-patterns

Agentic Design Patterns: Planning pattern



Source: https://machinelearningmastery.com/7-must-know-agentic-ai-design-patterns

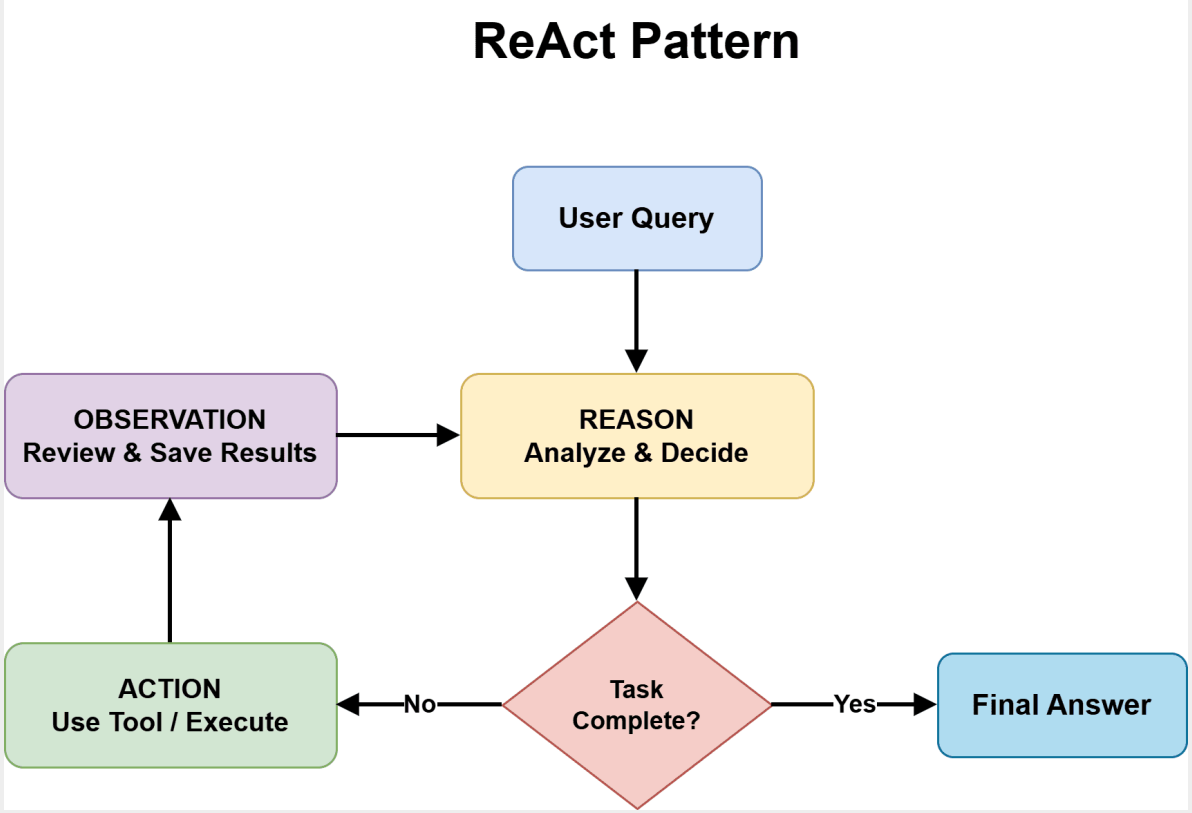
Agentic Design Patterns: Multi-agent pattern



Source: https://machinelearningmastery.com/7-must-know-agentic-ai-design-patterns

Agentic Design Patterns: ReAct

- design pattern for agentic systems
- builds on Chain-of-Thought prompting + tool-use
- simple but powerful



M.-T. Luong. "Neural Machine Translation", PhD Thesis, Stanford CS Dept., 2016

Exercise 4

- time to build our first agent!
- see `exercises/04_react.ipynb`
  - write from scratch
  - using langgraph

Evaluations

- evals: rigorous error analysis process

Evaluations

- evals: rigorous error analysis process
- **the** factor predicting success of an Agentic AI project

Evaluations

- evals: rigorous error analysis process
- **the** factor predicting success of an Agentic AI project
- requires a shift of mind set

Evaluations

- evals: rigorous error analysis process
- **the** factor predicting success of an Agentic AI project
- requires a shift of mind set
- deterministic vs. Probabilistic

Evaluations

- evals: rigorous error analysis process
- **the** factor predicting success of an Agentic AI project
- requires a shift of mind set
- deterministic vs. Probabilistic
- testing vs. evals

Evaluations

- evals: rigorous error analysis process
- **the** factor predicting success of an Agentic AI project
- requires a shift of mind set
- deterministic vs. Probabilistic
- testing vs. evals
- errors as inputs:
  - errors are valuable feedback used to help the agent recover and pivot

Agenda

1	Introduction	09:00
2	LLM basics	09.15
3	Agents	10.00
4	**Lab 1: Deep research**	11.00
5	Lunch	12.00
6	MCP	13.00
7	Agents: Advanced Concepts	14.00
8	Lab 2: MCP + advanced agents	15.15
9	Conclusion	16.45

Deep-research system

- GAIA is arguably the most comprehensive benchmark for agents
- Its questions are very difficult and hit on many challenges of LLM-based systems
- Here is an example of a hard question:

> Which of the fruits shown in the 2008 painting “Embroidery from Uzbekistan” were served as part of the October 1949 breakfast menu for the ocean liner that was later used as a floating prop for the film “The Last Voyage”? Give the items as a comma-separated list, ordering them in clockwise order based on their arrangement in the painting starting from the 12 o’clock position. Use the plural form of each fruit.

- OpenAI reported that GPT-4 only got 7% correct, while their deep-research system got 67%

Lab 1: Building a deep-research system

- we are ready for our first project
- check out the code at `labs/lab_01_deep_research.ipynb`
- Tasks:
  - code up a deep-research system
  - evaluate your system and compare with react agent
  - what works better?

Agenda

1	Introduction	09:00
2	LLM basics	09.15
3	Agents	10.00
4	Lab 1: Deep research	11.00
5	**Lunch**	12.00
6	MCP	13.00
7	Agents: Advanced Concepts	14.00
8	Lab 2: MCP + advanced agents	15.15
9	Conclusion	16.45

Agenda

1	Introduction	09:00
2	LLM basics	09.15
3	Agents	10.00
4	Lab 1: Deep research	11.00
5	Lunch	12.00
6	**MCP**	13.00
7	Agents: Advanced Concepts	14.00
8	Lab 2: MCP + advanced agents	15.15
9	Conclusion	16.45

Model Context Protocol (MCP)

- first released Nov 24

Model Context Protocol (MCP)

- first released Nov 24
- supports tools, resources and prompts
  - exetuble actions (API calls, running scripts)
  - local files, db queries, cloud documents etc.
  - structured prompts

Model Context Protocol (MCP)

- first released Nov 24
- supports tools, resources and prompts
  - exetuble actions (API calls, running scripts)
  - local files, db queries, cloud documents etc.
  - structured prompts
- recently donated to Linux Foundation: https://www.anthropic.com/news/donating-the-model-context-protocol-and-establishing-of-the-agentic-ai-foundation

Why use MCP?

- interoperability: solves the NxM problem

Why use MCP?

- interoperability: solves the NxM problem
- reusing toolsets across different agents with slightly different APIs

Why use MCP?

- interoperability: solves the NxM problem
- reusing toolsets across different agents with slightly different APIs
- not required for tool calling: list of available tools it provided to LLM with every request

Downsides of MCP

- slower than calling a tool locally (needs network request etc)

Downsides of MCP

- slower than calling a tool locally (needs network request etc)
- letting agents write code to compose MCP much more effective

Downsides of MCP

- slower than calling a tool locally (needs network request etc)
- letting agents write code to compose MCP much more effective
- and much more token efficient

Downsides of MCP

- slower than calling a tool locally (needs network request etc)
- letting agents write code to compose MCP much more effective
- and much more token efficient
- MCP offers no way for servers to declare their runtime/dependency needs

Downsides of MCP

- slower than calling a tool locally (needs network request etc)
- letting agents write code to compose MCP much more effective
- and much more token efficient
- MCP offers no way for servers to declare their runtime/dependency needs
- Since tools are drawn from arbitrary sources, they are not aware of what other tools are available to the agent. Their instructions can’t account for the rest of the toolbox

Downsides of MCP

- slower than calling a tool locally (needs network request etc)
- letting agents write code to compose MCP much more effective
- and much more token efficient
- MCP offers no way for servers to declare their runtime/dependency needs
- Since tools are drawn from arbitrary sources, they are not aware of what other tools are available to the agent. Their instructions can’t account for the rest of the toolbox
- Agents tend to be less effective at tool use as the number of tools grows

Exercise 5

- see `exercises/05_mcp.ipynb`
  - write MCP server
  - connect to MCP server

Dynamic MCP

- one problem with MCP: too many tools call overwhelm the model
- fixed toolset
- instead: https://www.docker.com/blog/dynamic-mcps-stop-hardcoding-your-agents-world/

Workflow	Before: Static MCP setup	After: Dynamic MCPs	Impact
Tool discovery	Manually browse the MCP servers	mcp-find searches a Docker MCP Catalog (230+ servers)	Faster discovery
Adding tools	Enable the MCP servers manually	mcp-add pulls only the servers an agent needs	Zero manual config; just-in-time tooling
Authentication	Configure the MCP servers ahead of time	Prompt user to complete OAuth when required	Smoother onboarding flows (mcp-ui)
Tool composition	Agent generated tool calls; definitions sent to model	With code-mode, agents write code using multiple tools	Multi-tool workflows and unified outputs
Context size	Load lots of unused tool definitions	Keep only the tools actually required for the task	Lower token usage and latency
Future-proofing	Static integrations	Dynamic, composable tools with sandboxed scripting	Ready for evolving agent behaviors
Developer involvement	Constant context switching and config hacking	Agents self-serve: discover, authorize, and orchestrate	Fewer manual steps; better focus time

Agenda

1	Introduction	09:00
2	LLM basics	09.15
3	Agents	10.00
4	Lab 1: Deep research	11.00
5	Lunch	12.00
6	MCP	13.00
7	**Agents: Advanced Concepts**	14.00
8	Lab 2: MCP + advanced agents	15.15
9	Conclusion	16.45

Advanced Concepts: Coding agents

- LLMs are good at code. Why must tool-calls be written in JSON?

Advanced Concepts: Coding agents

- LLMs are good at code. Why must tool-calls be written in JSON?
- https://blog.cloudflare.com/code-mode/:

> “[…] We tried something different: Convert the MCP tools into a TypeScript API, and then ask an LLM to write code that calls that API. […] We found agents are able to handle many more tools, and more complex tools, when those tools are presented as a TypeScript API rather than directly.” […] The approach really shines when an agent needs to string together multiple calls. With the traditional approach, the output of each tool call must feed into the LLM’s neural network, just to be copied over to the inputs of the next call, wasting time, energy, and tokens. When the LLM can write code, it can skip all that, and only read back the final results it needs.

Advanced Concepts: Coding agents

- LLMs are good at code. Why must tool-calls be written in JSON?
- https://blog.cloudflare.com/code-mode/:

> "[…] We tried something different: Convert the MCP tools into a TypeScript API, and then ask an LLM to write code that calls that API. […] We found agents are able to handle many more tools, and more complex tools, when those tools are presented as a TypeScript API rather than directly." […] The approach really shines when an agent needs to string together multiple calls. With the traditional approach, the output of each tool call must feed into the LLM's neural network, just to be copied over to the inputs of the next call, wasting time, energy, and tokens. When the LLM can write code, it can skip all that, and only read back the final results it needs.

- `smolagents` library is based on this pattern

Code agent

Instruction: Determine the most cost-effective country to purchase the smartphone model "CodeAct 1". The countries to consider are the USA, Japan, Germany, and India.

Available APIs
[1] lookup_rates(country: str) -> (float, float)
[2] convert_and_tax(price: float, exchange_rate: float, tax_rate: float) -> float
[3] estimate_final_price(converted_price: float, shipping_cost: float) -> float
[4] lookup_phone_price(model: str, country: str) -> float
[5] estimate_shipping_cost(destination_country: str) -> float

LLM Agent using [**Text/JSON**] as Action

Think I should calculate the phone price in USD for each country, then find the most cost-effective country.

Action
```
Text: lookup_rates, Germany
JSON: {"tool": "lookup_rates", "country": "Germany"}
```

Environment 1.1, 0.19

Action
```
Text: lookup_phone_price, CodeAct 1, Germany
JSON: {"tool": "lookup_phone_price", "model": "CodeAct 1", "country": "Germany"}
```

Environment 700

Action
```
Text: convert_and_tax, 700, 1.1, 0.19
JSON: {"tool": "convert_and_tax", "price": 700, "exchange_rate": 1.1, "tax_rate": 0.19}
```

Environment 916.3

[... interactions omitted (look up shipping cost and calculate final price) ...]

Action
```
Text: lookup_rates, Japan
JSON: {"tool": "lookup_rates", "country": "Japan"}
```

[... interactions omitted (calculate final price for all other countries)...]

Response The most cost-effective country to purchase the smartphone model is Japan with price 904.00 in USD.

> **Fewer Actions Required!**

CodeAct: LLM Agent using [**Code**] as Action

Think I should calculate the phone price in USD for each country, then find the most cost-effective country.

Action
```
countries = ['USA', 'Japan', 'Germany', 'India']
final_prices = {}

for country in countries:
    exchange_rate, tax_rate = lookup_rates(country)
    local_price = lookup_phone_price("xAct 1", country)
    converted_price = convert_and_tax(
      local_price, exchange_rate, tax_rate
    )
    shipping_cost = estimate_shipping_cost(country)
    final_price = estimate_final_price(converted_price, shipping_cost)
    final_prices[country] = final_price

most_cost_effective_country = min(final_prices, key=final_prices.get)
most_cost_effective_price = final_prices[most_cost_effective_country]
print(most_cost_effective_country, most_cost_effective_price)
```

Control & Data Flow of Code Simplifies Complex Operations

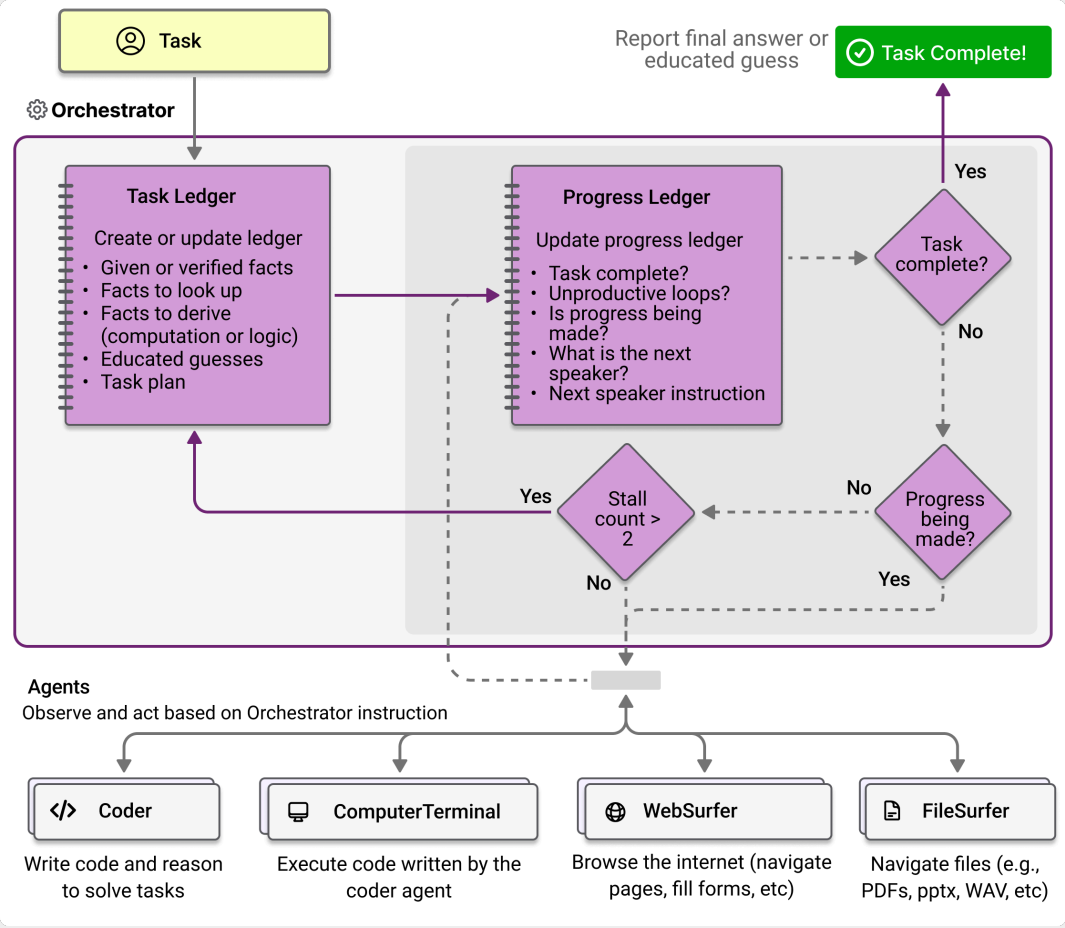
Re-use `min` Function from Existing Software Infrastructures (Python library)

Environment 1.1, 0.19

Response The most cost-effective country to purchase the smartphone model is Japan with price 904.00 in USD.

Source: https://huggingface.co/blog/open-deep-research#the-gaia-benchmark

Magentic One



Source: https://www.microsoft.com/en-us/research/articles/magentic-one-a-generalist-multi-agent-system-for-solving-complex-tasks/

Advanced Concepts: Memory

- in principle, memory allows the agent to self-improve

Advanced Concepts: Memory

- in principle, memory allows the agent to self-improve
- in practice, no one knows yet how to best make use of it

Advanced Concepts: Memory

- in principle, memory allows the agent to self-improve
- in practice, no one knows yet how to best make use of it
- https://github.com/letta-ai/letta

Advanced Concepts: Memory

- in principle, memory allows the agent to self-improve
- in practice, no one knows yet how to best make use of it
- https://github.com/letta-ai/letta
- again, we can use tools:
  - create memory after learning something
  - recall memories given some query

Exercise 06: Code execution and memory

- see `exercises/06_code_execution.ipynb`
- Goals:
  - build a simple data analysis agent
  - give the agent memory

Exercise 07: Code agent pattern

- see `exercises/07_coding-agent.ipynb`

The Risk: “What is the worst that can happen?”

Even with Docker, a malicious or confused agent with root inside a container can cause damage if not properly sandboxed:

- Host Filesystem Wipe: If mounting `-v ./data:/data` and the agent runs `rm -rf /data/*`: this deletes the files on your actual laptop/server

The Risk: “What is the worst that can happen?”

Even with Docker, a malicious or confused agent with root inside a container can cause damage if not properly sandboxed:

- Host Filesystem Wipe: If mounting `-v ./data:/data` and the agent runs `rm -rf /data/*`: this deletes the files on your actual laptop/server
- Network Attacks: The agent can use curl or Python to scan your local network (192.168.1.x), attack other devices, or access internal services (like a local database) that have no password.

The Risk: “What is the worst that can happen?”

Even with Docker, a malicious or confused agent with root inside a container can cause damage if not properly sandboxed:

- Host Filesystem Wipe: If mounting `-v ./data:/data` and the agent runs `rm -rf /data/*`: this deletes the files on your actual laptop/server
- Network Attacks: The agent can use curl or Python to scan your local network (192.168.1.x), attack other devices, or access internal services (like a local database) that have no password.
- Resource Exhaustion (DoS): The agent could launch a “fork bomb” or fill the disk, crashing your host machine.

The Risk: “What is the worst that can happen?”

Even with Docker, a malicious or confused agent with root inside a container can cause damage if not properly sandboxed:

- Host Filesystem Wipe: If mounting `-v ./data:/data` and the agent runs `rm -rf /data/*`: this deletes the files on your actual laptop/server
- Network Attacks: The agent can use curl or Python to scan your local network (192.168.1.x), attack other devices, or access internal services (like a local database) that have no password.
- Resource Exhaustion (DoS): The agent could launch a “fork bomb” or fill the disk, crashing your host machine.
- Container Escape (Rare): If there is a kernel vulnerability, running as root increases the chance of “escaping” the container to control the host OS.

The Risk: “Lethal trifecta”

1. Access to private data

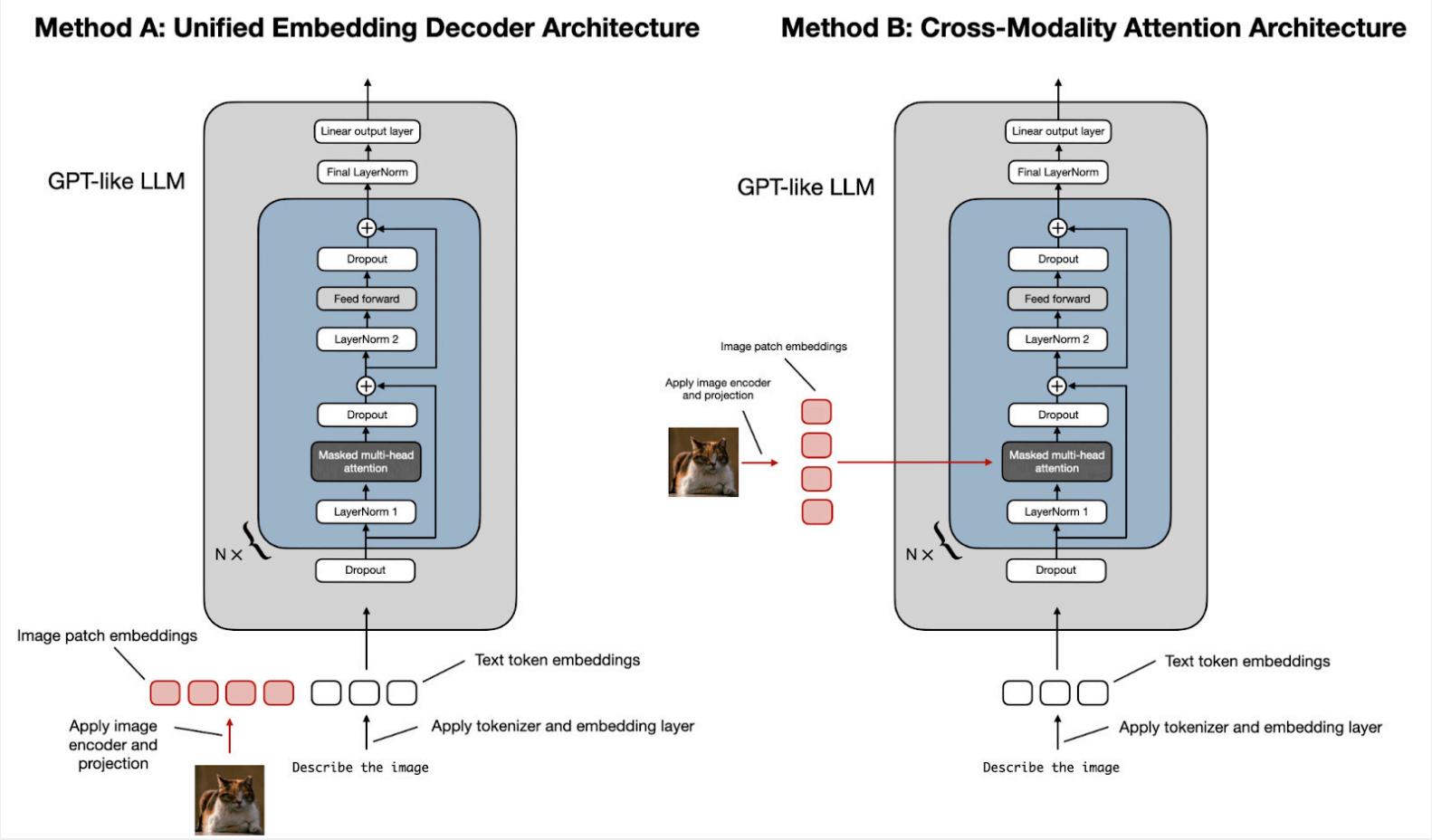
The Risk: “Lethal trifecta”

1. Access to private data
2. Ability to externally communicate

The Risk: “Lethal trifecta”

1. Access to private data
2. Ability to externally communicate
3. Exposure to untrusted content

Multimodal LLMs



Source: https://magazine.sebastianraschka.com/p/understanding-multimodal-llms

Exercise 08: Image Understanding

- see `exercises/08_image_understanding.ipynb`

Advanced Concepts: Deep Agents

- Agents can increasingly tackle long-horizon tasks
  - agent task length doubling every 7 months

Advanced Concepts: Deep Agents

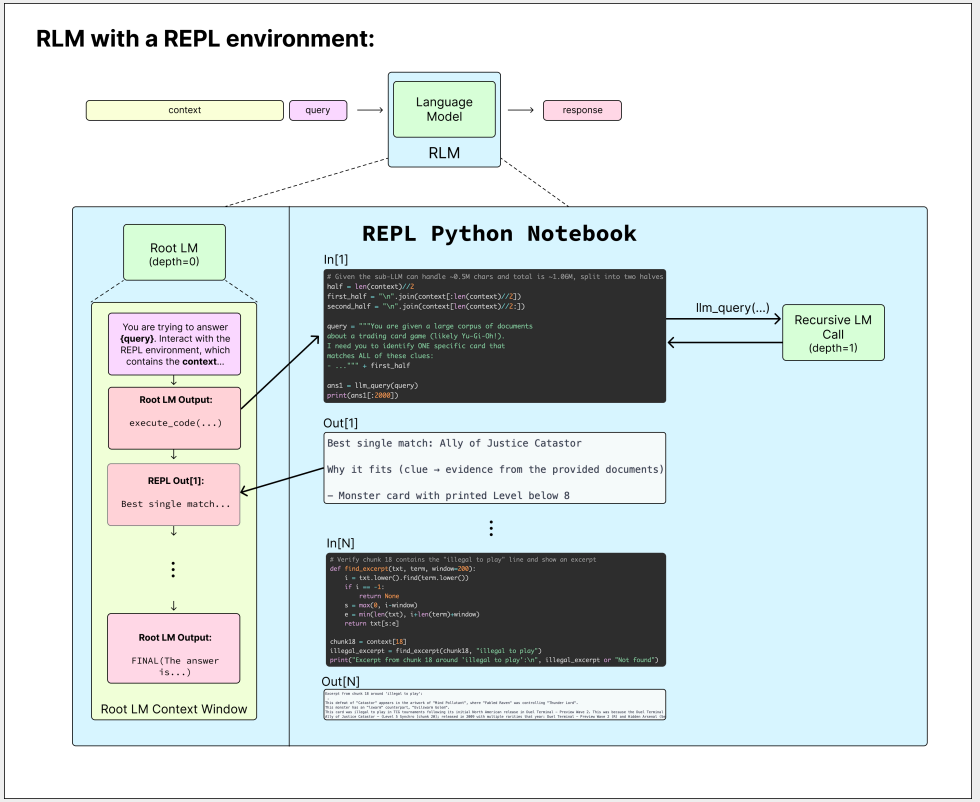
- Agents can increasingly tackle long-horizon tasks
  - agent task length doubling every 7 months
- But: long horizon tasks often span dozens of tool calls, which present cost and reliability challenges.

Advanced Concepts: Deep Agents

- Agents can increasingly tackle long-horizon tasks
  - agent task length doubling every 7 months
- But: long horizon tasks often span dozens of tool calls, which present cost and reliability challenges.
- Popular agents such as Claude Code and Manus use some common principles to address these challenges, including:
  - planning (prior to task execution)
  - computer access (giving the agent access to a shell and a filesystem)
  - sub-agent delegation (isolated task execution)

Advanced Concepts: RLM

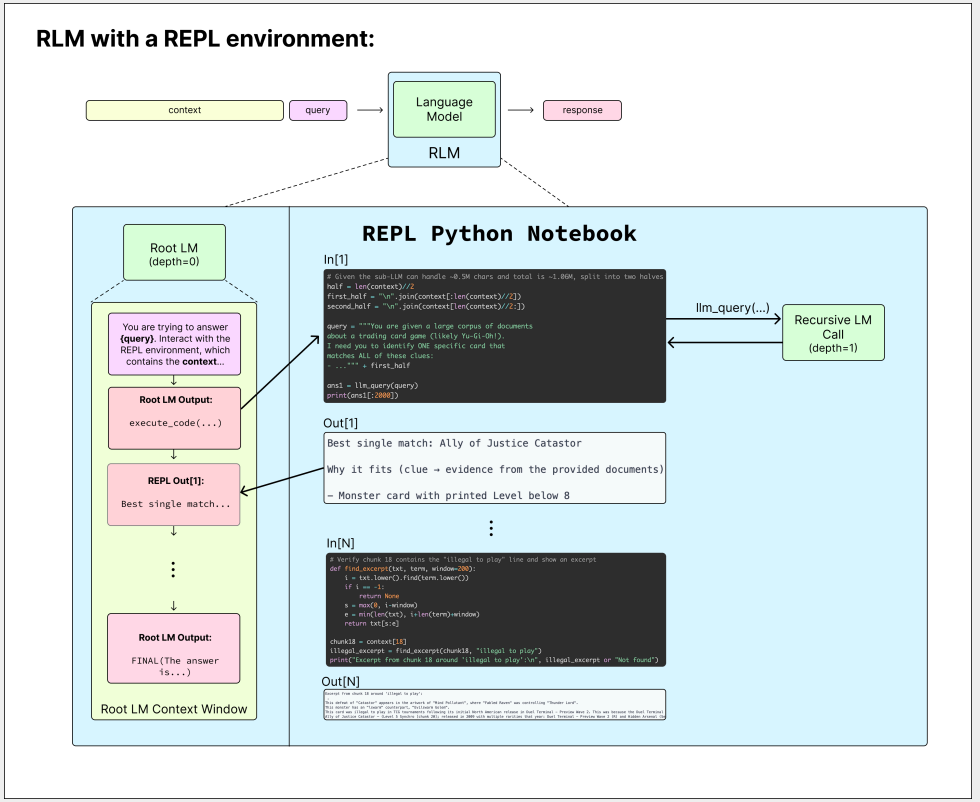
- Another recent innovation: Recursive Language Models (RLMs)



"Source: https://alexzhang13.github.io/blog/2025/rlm/"

Advanced Concepts: RLM

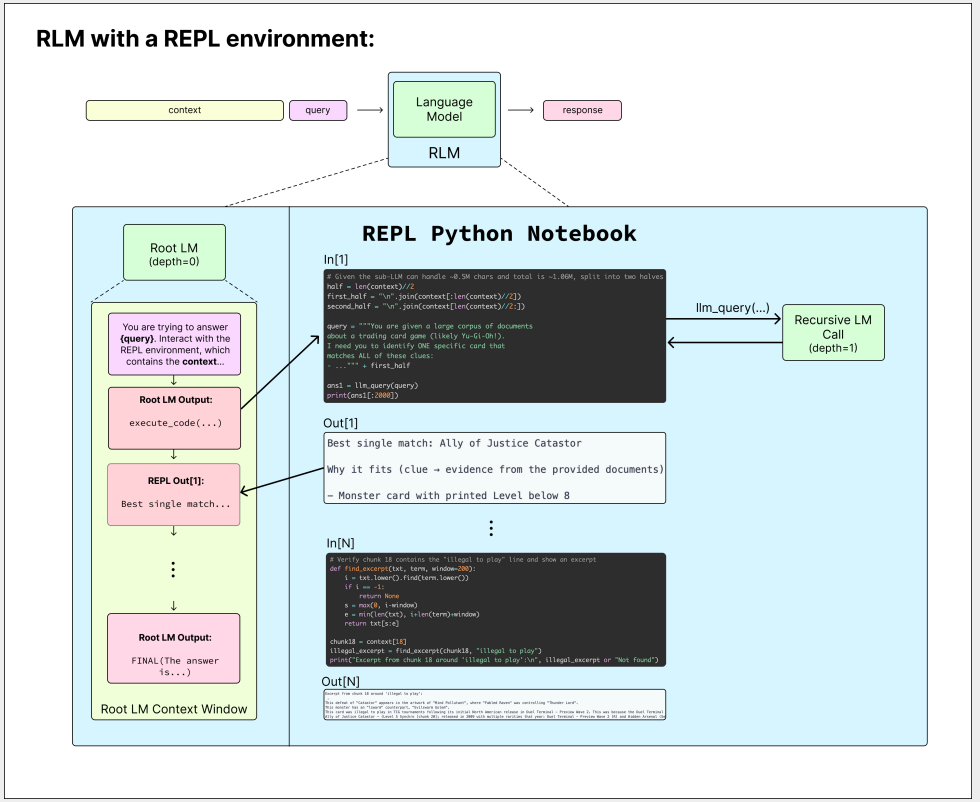
- Another recent innovation: Recursive Language Models (RLMs)
- RLM provides the illusion of near infinite context



"Source: https://alexzhang13.github.io/blog/2025/rlm/"

Advanced Concepts: RLM

- Another recent innovation: Recursive Language Models (RLMs)
- RLM provides the illusion of near infinite context
- under the hood a language model manages, partitions, and recursively calls itself or another LM over the context accordingly to avoid context rot



"Source: https://alexzhang13.github.io/blog/2025/rlm/"

Agenda

1	Introduction	09:00
2	LLM basics	09.15
3	Agents	10.00
4	Lab 1: Deep research	11.00
5	Lunch	12.00
6	MCP	13.00
7	Agents: Advanced Concepts	14.00
8	**Lab 2: MCP + advanced agents**	15.15
9	Conclusion	16.45

Lab 2: Building a deep-agent with MCP

- see the starter code at `labs/lab_02_advanced.ipynb`
- Tasks:
  - implement a MAS with coding and web-search sub-agents
  - use the “progressive disclosure” pattern
  - evaluate your system
- Bonus: Allow the system “to see”.
- Bonus: Try evaluating against hard GAIA questions.

Agenda

1	Introduction	09:00
2	LLM basics	09.15
3	Agents	10.00
4	Lab 1: Deep research	11.00
5	Lunch	12.00
6	MCP	13.00
7	Agents: Advanced Concepts	14.00
8	Lab 2: MCP + advanced agents	15.15
9	**Conclusion**	16.45

Conclusion

- Some fun: https://gricha.dev/blog/the-highest-quality-codebase

Conclusion

- Some fun: https://gricha.dev/blog/the-highest-quality-codebase
- Future trends:
  - David Silver:
    - “models will be trained with RL for computer use”
    - “with this the models will go beyond human data: era of experience”

Conclusion

- Some fun: https://gricha.dev/blog/the-highest-quality-codebase
- Future trends:
  - David Silver:
    - “models will be trained with RL for computer use”
    - “with this the models will go beyond human data: era of experience”
- Feedback:
  - What was good? What can be improved?
  - What was missing? What was too much?