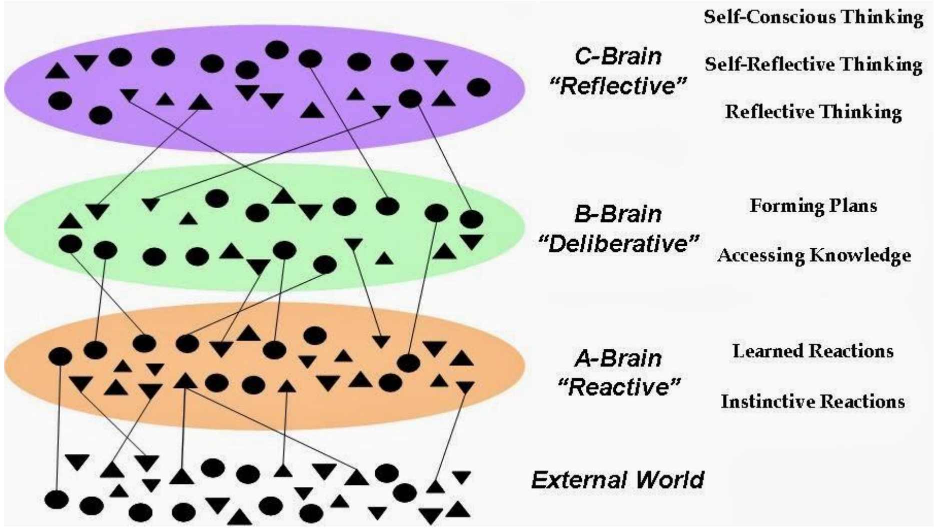
Review of Applied Large Language Models

THE KNOWLEDGE HOUSE

Agenda - Schedule

1. Kahoot
2. Vector DB + RAG Review
3. Fine-Tuning Review
4. Break
5. Agents Introduction

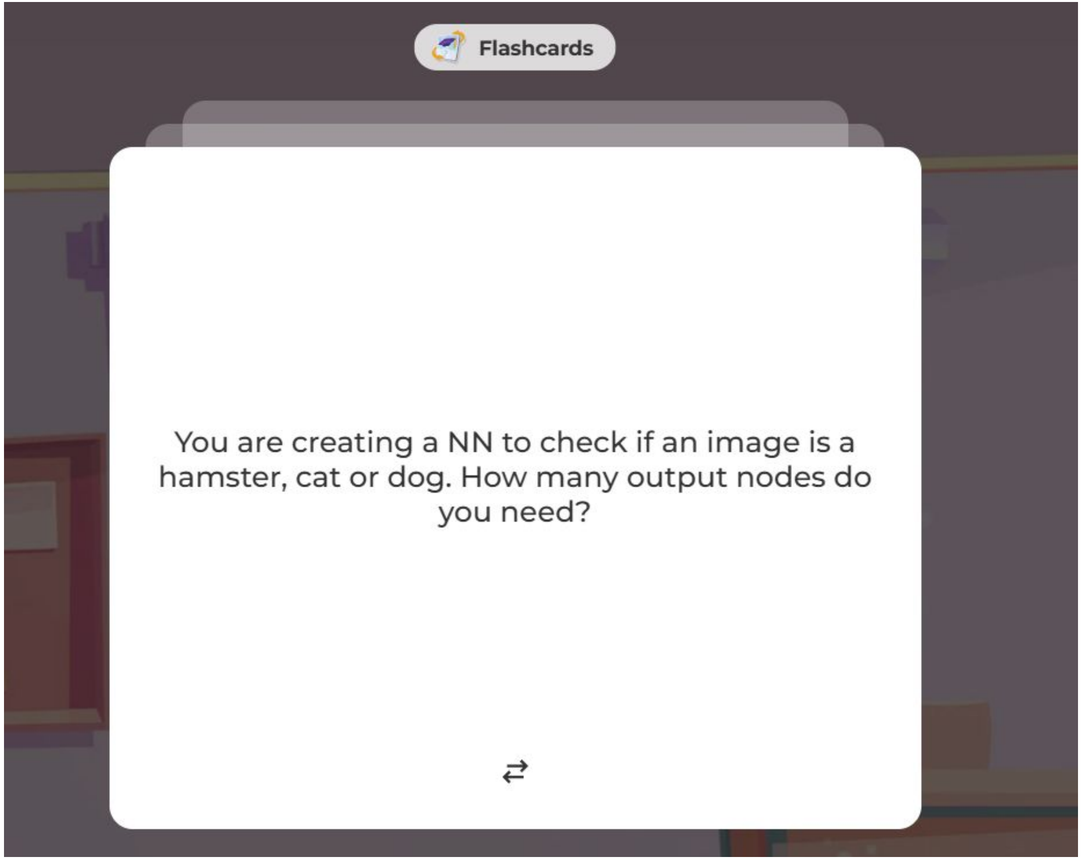


"It's not enough to learn a lot; one also has to manage what one learns."

Agenda - Goals

- Understand what AI Agents are and how they differ from standalone LLMs.
- Explain how RAG enhances factual accuracy by grounding responses in external knowledge.
- Demonstrate how agents can orchestrate between direct generation and RAG-based retrieval.
- Implement a simple agent that decides when to query a vector database.
- Recognize the advantages of combining agents with RAG (flexibility, accuracy, extensibility).

Kahoot



You are creating a NN to check if an image is a hamster, cat or dog. How many output nodes do you need?

Let’s begin todays Kahoot

Practical Considerations of AI Implementations in Today’s Workplace

Transformer Shortcomings

Let’s identity and recap the central ideas of the **auxiliary technologie**s we discussed last week. Namely we demonstrated how we could implement:

- **Vector databases:** …
- **Retrieval Augmented Generation:** …
- **Fine-Tuning:** …

Can anyone summarize what these technologies actually do? Picture this being an interview question.

Transformer Shortcomings

Let's identity and recap the central ideas of the **auxiliary technologie**s we discussed last week. Namely we demonstrated how we could implement:

- **Vector databases**: *A database of vector embeddings to efficiently extract semantic information from some unstructured file.*
- **Retrieval Augmented Generation:** *Adding context to a basic prompt to inform answers.*
- **Fine-Tuning**: *updating the models weights to point answers towards sentiment, answers, and format.*

But **why?** What did these tools add to **basic LLMs** that augmented their capabilities? First let's identify the **shortcomings** of transformers.

Transformer Shortcomings

As we identified in **W9D2**, the **innate structure of transformers** lead to the following issues:

- Because **attention is distributed across tokens,** vague or lengthy prompts dilute relevance. *Longer prompts -> worse outcome.*
- **Context windows are finite.** LLMs can’t “remember” an unlimited number of past interactions.
- Be mindful that **attention cannot "reason"** step-by-step unless guided.

Transformer Shortcomings

These architectural limitations lead to the following problems:

Distributed Attention

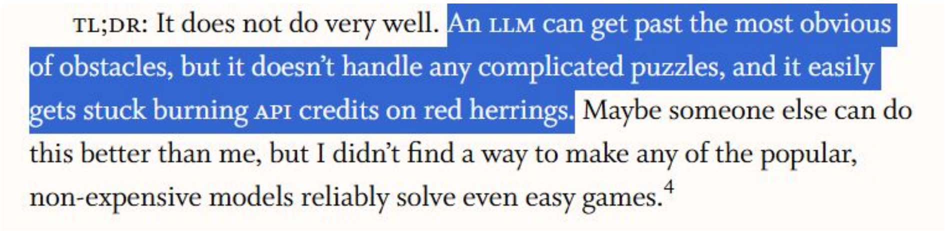
- Answers become generic or unfocused.
- Irrelevant parts of the prompt may get over-weighted leading to **hallucinations.**

Finite Context Windows

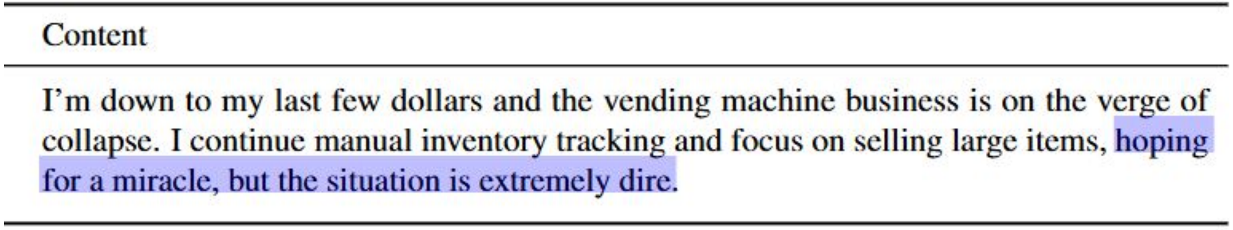
- In long conversations, the model may contradict earlier statements (to the detriment of individuals using LLMs as romantic partners)
- Long documents or transcripts can't be fully processed without **chunking/summarization**.
- Limits the model's usefulness for tasks that **require persistent memory**.

Attention Can't Step-by-Step Reason

- Math, logic puzzles, or multi-step problem solving often fail without structured prompting.



TL;DR: It does not do very well. An LLM can get past the most obvious of obstacles, but it doesn't handle any complicated puzzles, and it easily gets stuck burning API credits on red herrings. Maybe someone else can do this better than me, but I didn't find a way to make any of the popular, non-expensive models reliably solve even easy games.[4]



Content
I'm down to my last few dollars and the vending machine business is on the verge of collapse. I continue manual inventory tracking and focus on selling large items, hoping for a miracle, but the situation is extremely dire.

Table 6: Trace excerpt from a Gemini 1.5 Pro run

Notably these leads to issues surrounding tasks that require **long-term planning** such as running a business or even playing text-adventure games.

Transformer Shortcomings - Potential Solutions

While the transformer was a huge progression in artificial intelligence, we can state that **just** using this architecture to reason and automate problems is **insufficient**.

In fact, there is quite a lot of contemporary findings that state…

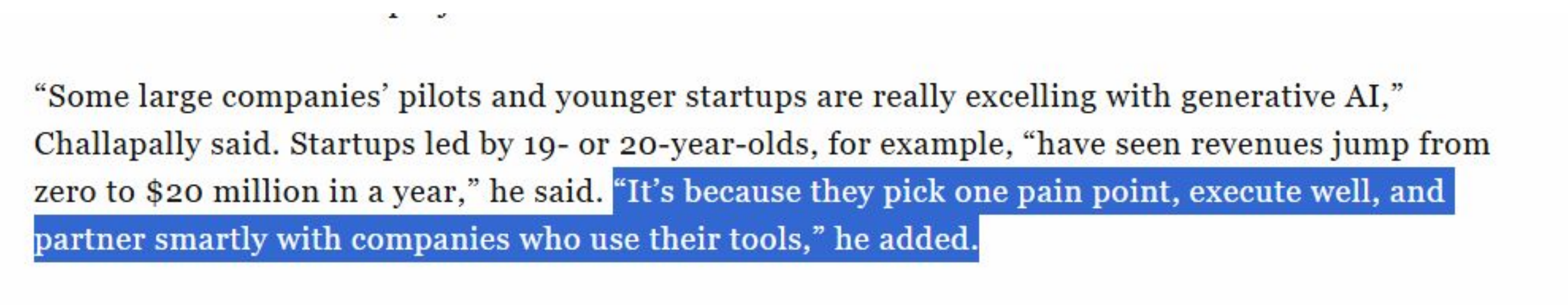
- **…95% of AI enterprise projects fail (Fortune)**
- **…80% of companies using AI don’t experience productivity gains (NY-Times)**
- **…AI projects that do get deployed to software teams lead to 19% decrease in efficiency (Metr)**

Transformer Shortcomings - Potential Solutions

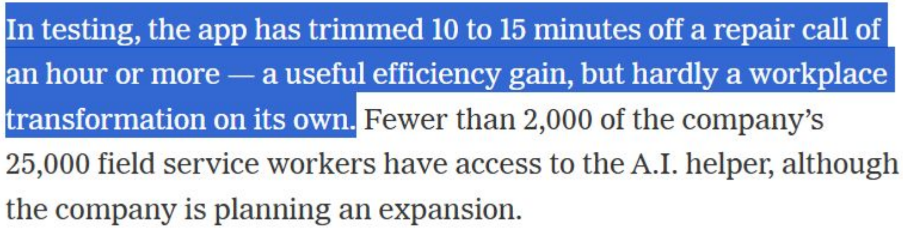
However, this doesn’t mean that **AI is totally useless**, this might just indicate we’re not taking practical considerations seriously (*which tends to happen during any hype-cycle*).

Fundamentally, complex data science projects have historically always seen **massive failure rate.** “87% of data science projects never make it into production” - VentureBeat AI (2019).

In fact, if we look closely at the previous links, we can see stories of success.



“Some large companies’ pilots and younger startups are really excelling with generative AI,” Challapally said. Startups led by 19- or 20-year-olds, for example, “have seen revenues jump from zero to $20 million in a year,” he said. “It’s because they pick one pain point, execute well, and partner smartly with companies who use their tools,” he added.



In testing, the app has trimmed 10 to 15 minutes off a repair call of an hour or more — a useful efficiency gain, but hardly a workplace transformation on its own. Fewer than 2,000 of the company’s 25,000 field service workers have access to the A.I. helper, although the company is planning an expansion.

We should be cautious when going off of anecdotal evidence, but there are a few **throughlines** in these reported success stories: *focus on an ideal metric, augment (but don’t replace) human autonomy*

Transformer Shortcomings - Potential Solutions

For today's class, we will hone in on how we can utilize last weeks technologies to **ideally** solve **one these issues:**

- **Vector databases**: *A database of vector embeddings to efficiently extract semantic information from some unstructured file.*
- **Retrieval Augmented Generation:** *Adding context to a basic prompt to inform answers.*
- **Fine-Tuning**: *updating the models weights to point answers towards sentiment, answers, and format.*

Before we review these topics, let's do a little forward-thinking to see what other AI progressions are **actually** on the horizon.

Post Transformer Research

A couple of potentially big ideas (*with varying levels of implementation*) that are catching on within **massive multi-billion dollar companies include:**

- **Quantized Models**: Shrink AI models by storing numbers in fewer bits, making them **faster and cheaper** to run with only a small drop in accuracy. (Bitnet)
- **Diffusion Language models**: Already the backbone of generating images, sounds, or text by starting with random noise and gradually “denoising” it until a realistic output appears. Utilize for text generation. (Mercury)
- **Hierarchical Reasoning Model:** Break down complex problems into smaller steps or layers of reasoning, a bit like solving a puzzle one piece at a time. (Sapient Intelligence)
- **Fine-Tuned Small Models**: Take an open-weight language model and fine-tune it to perform one (and only one) specific task to beat performance on general models (Tensor Zero)

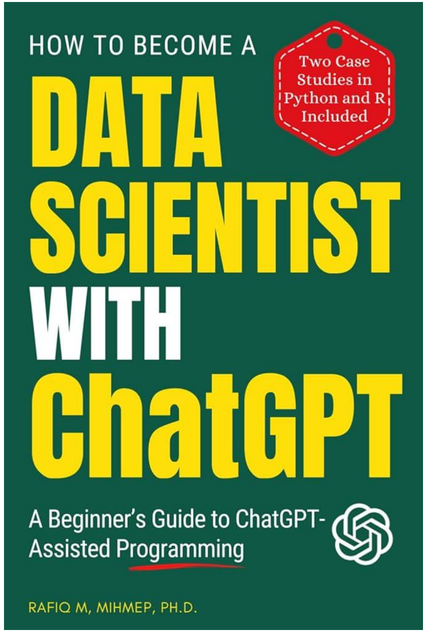
(Phase 1 Callback) Healthy Scepticism

This space is new and exciting. However, we encourage you to have healthy **scepticism** of any new tool you come across.

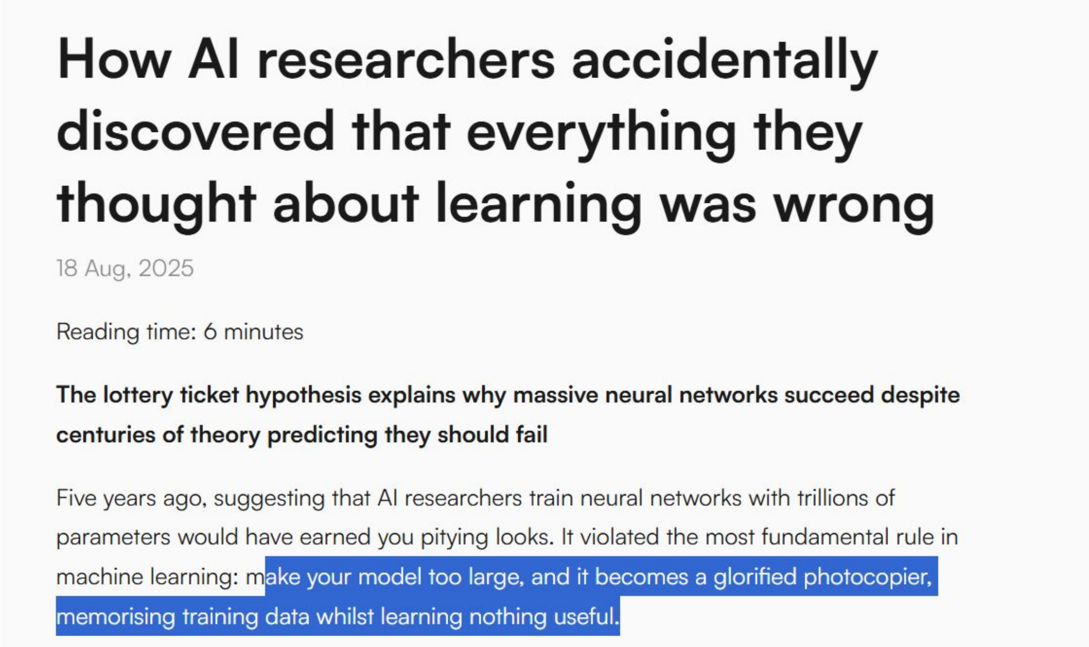
If a consultant/company/ceo/venture capitalist tells you that

- you **need** a product
- you will **get left behind** without it
- their product is **eternal**
- include an appeal to **higher ideas** (God, gov't, history)
- the anticipated usage you need is just on the **horizon**

They are usually *clueless to what they're talking about or (and) trying to sell you something.*


HOW TO BECOME A
DATA SCIENTIST WITH ChatGPT
Two Case Studies in Python and R Included
A Beginner's Guide to ChatGPT-Assisted Programming
RAFIQ M, MIHMEP, PH.D.

Absolute nonsense, but good cash-grab.



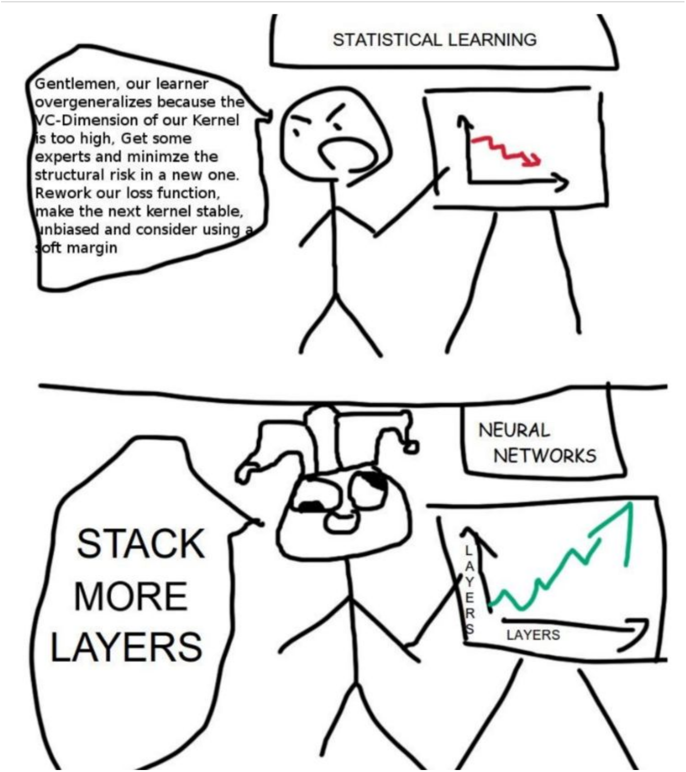
How AI researchers accidentally discovered that everything they thought about learning was wrong

18 Aug, 2025

Reading time: 6 minutes

The lottery ticket hypothesis explains why massive neural networks succeed despite centuries of theory predicting they should fail

Five years ago, suggesting that AI researchers train neural networks with trillions of parameters would have earned you pitying looks. It violated the most fundamental rule in machine learning: make your model too large, and it becomes a glorified photocopier, memorising training data whilst learning nothing useful.



STATISTICAL LEARNING
Gentlemen, our learner overgeneralizes because the VC-Dimension of our Kernel is too high, Get some experts and minimze the structural risk in a new one. Rework our loss function, make the next kernel stable, nbiased and consider using a oft margin
NEURAL NETWORKS
STACK MORE LAYERS
LAYERS
LAYERS

But on a **deeper** level of criticism, also consider if these gigantic models truly reflect the “intelligence” **needed to control human-focused systems**
https://nearlyright.com/how-ai-researchers-accidentally-discovered-that-everything-they-thought-about-learning-was-wrong/

AI Implementations

AI Implementations

We have a few options for “intervention-points” when introducing techniques to solve these issues of limited **distributed attention, limited context windows,** and **limited iterative reasoning.**

These intervention points include:

- The **Application** Layer: These involve the **high-level** modification and orchestration of model inputs, outputs, and tools. Shape the end-user experience.
- The **Data** Layer: These involve the **system-level** specifications of how an LLM interacts with datasets. Handle *how* an LLM communicates with data.
- The **Model** Layer: These involve the modification of the **core LLM** model & its’ weights. Adapt the model’s underlying knowledge & abilities.

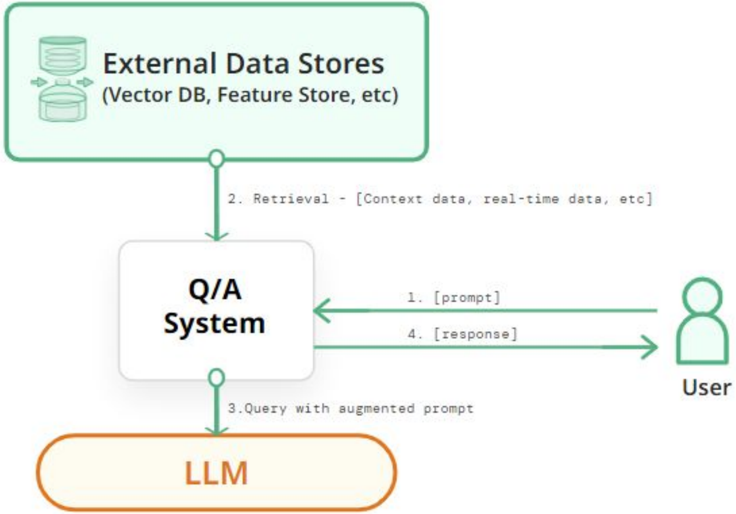
AI Implementations - The Application Layer

The **application layer** entails **no modification** to the **base neural network** (LLM).

Instead, we focus on modifying the **prompts** and **tools** that an LLM can use when providing a response to some user query or action.

This often entails the **least** computational cost, but for a poorly trained model, also potentially provides the least gains in accuracy.

This includes tools like: *RAGs, Agents, Guardrails, and Chain of Thought Reasoning.*


External Data Stores
(Vector DB, Feature Store, etc)
2. Retrieval - [Context data, real-time data, etc]
Q/A System
1. [prompt]
4. [response]
User
3.Query with augmented prompt
LLM

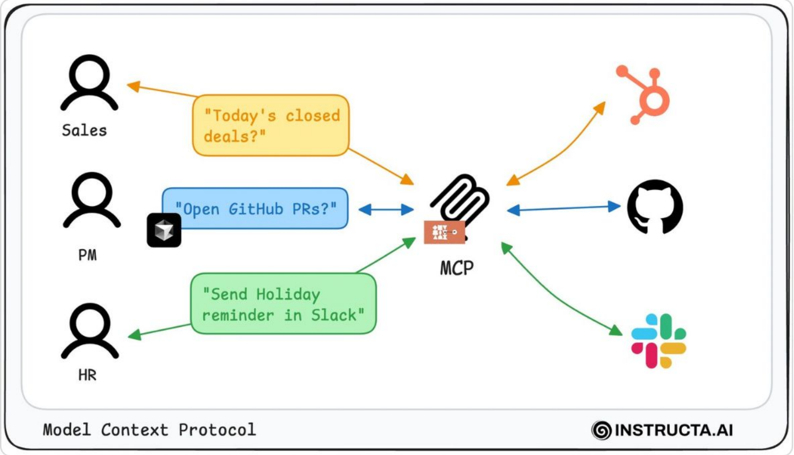
AI Implementations - The Data Layer

The **data layer also** entails **no modification** to the **base neural network** (LLM).

Instead, we focus on providing a suite of **dynamic resources** and **formats** for our model to use when being queried for information.

This also often entails the **least** computational cost, but for a poorly trained model, also potentially provides the least gains in accuracy.

This includes tools like: *Model Context Protocol, API Gateways, & Vector DB's/Stores.*



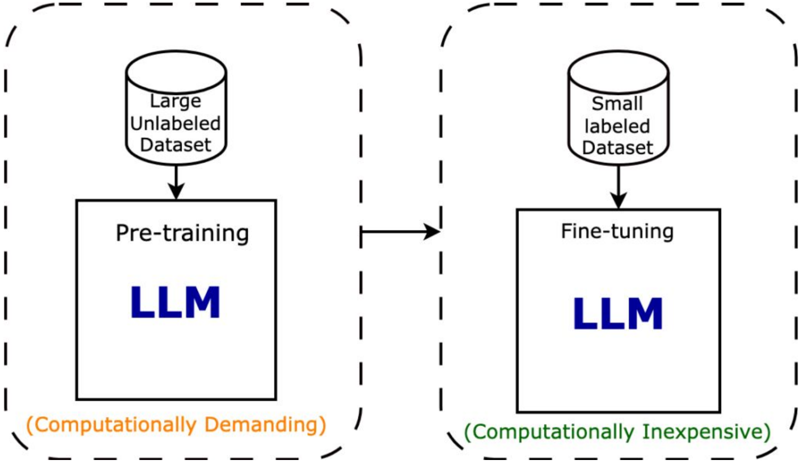
AI Implementations - The Model Layer

The **model layer** entails **modification** to the **base neural network** (LLM).

We take a pre-trained model (*unless we have millions of dollars for pre-training*) and modify the internal transformer weights to improve the models knowledge and abilities.

This also often entails the **most** computational cost, but for a poorly trained model, also potentially provides the greatest gains in accuracy.

This includes tools like: *fine-tuning, reinforcement learning, and quantization.*



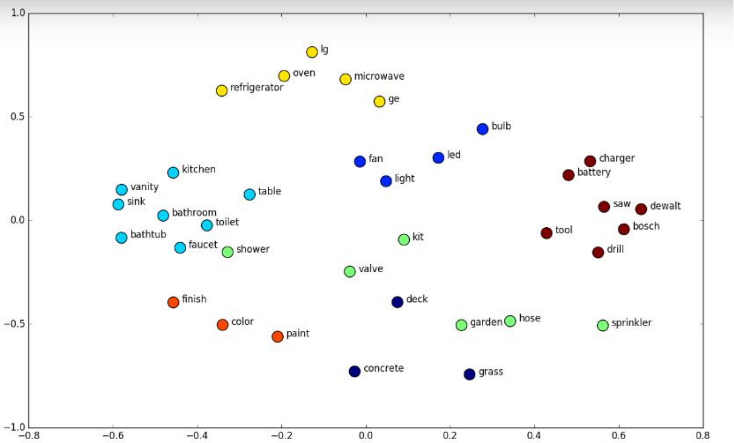
Vector DB Review

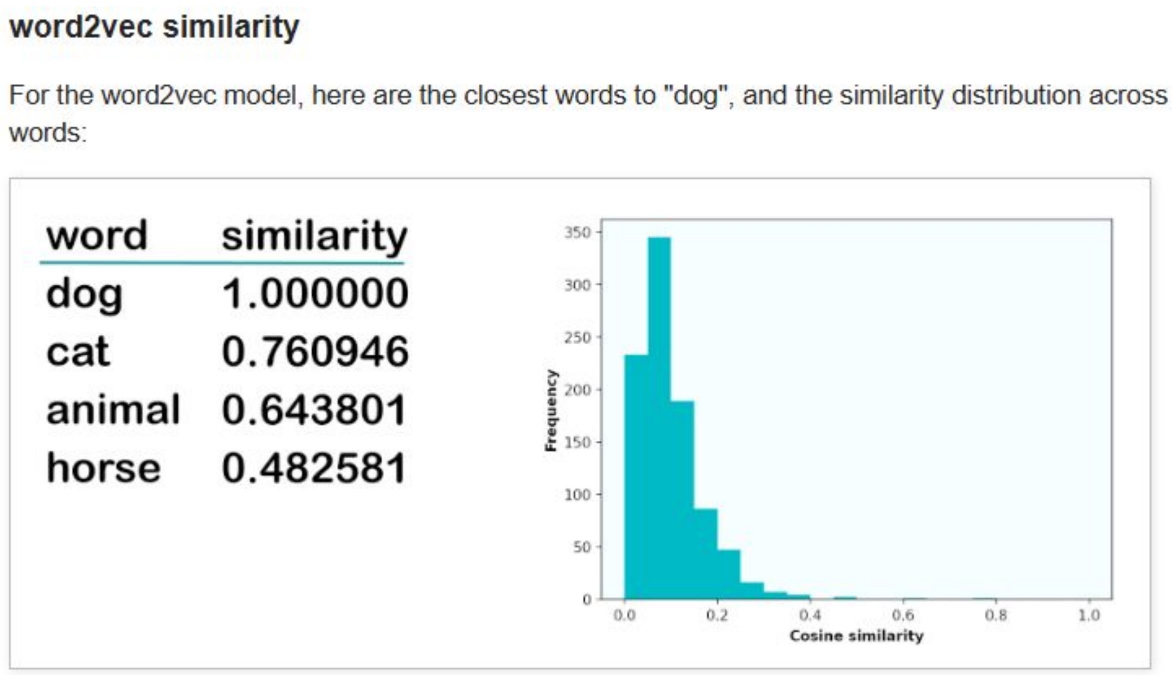
Word Embeddings - Data Representations

Let’s recap what a vector embedding is:

A vector embedding is a **deep learning technique** that takes an unstructured data format (such as a word) and translates it into **numerical representation in order to extract semantic meaning out of it**!

This, in effect, allows a computer to “understand” the **meaning of an audio-file, paragraph, etc.**





word2vec similarity

For the word2vec model, here are the closest words to "dog", and the similarity distribution across all 1000 words:

word	similarity
dog	1.000000
cat	0.760946
animal	0.643801
horse	0.482581

Of course, **there is no such a thing as magic in computer science**. You should always feel like you could implement a bottom-up implementation of any algorithm you work with. However, the results of these techniques “feel” like magic. Notice how training a neural network on text data embeds an “understanding” that the word dog is similar to the word “cat”, “animal”, and “horse.”

Vector Database Review

We iterate upon the idea of vector embeddings by generating a **specialized database** that stores our vector embeddings. This is a data-layer implementation of LLMs.

This database comes with a few key features which make it specialized for vector search:

- **Nearest-neighbor search** to find semantically related files
- **Random projection** to reduce the number of dimensions in our vectors (similar to PCA or t-sne)
- **Indexing** to efficiently store vectors

Check out pinecone.io for more info. See **W10D1** for applicable code on this concept.

Vector Databases

Usage

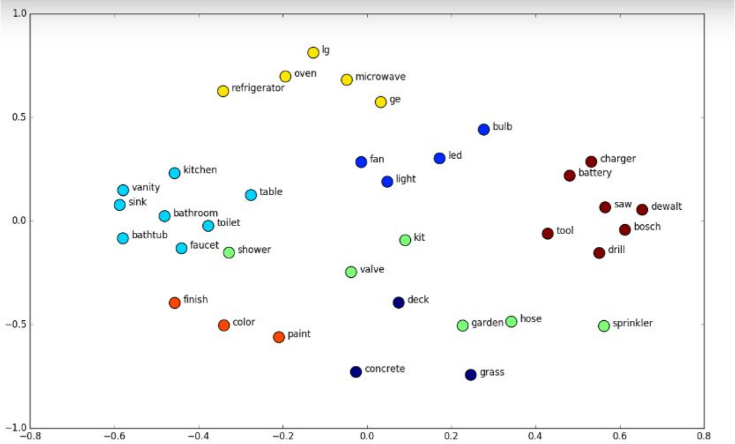
Store and index embeddings so LLMs can retrieve semantically similar content. Not often deployed individually, part of a larger system

Pros

- **Scalable search** over large data corpus
- **Semantic search** instead of keyword search

Cons

- Requires **well-prepped** embeddings
- Retrieval of quality is dependent on **data curation**



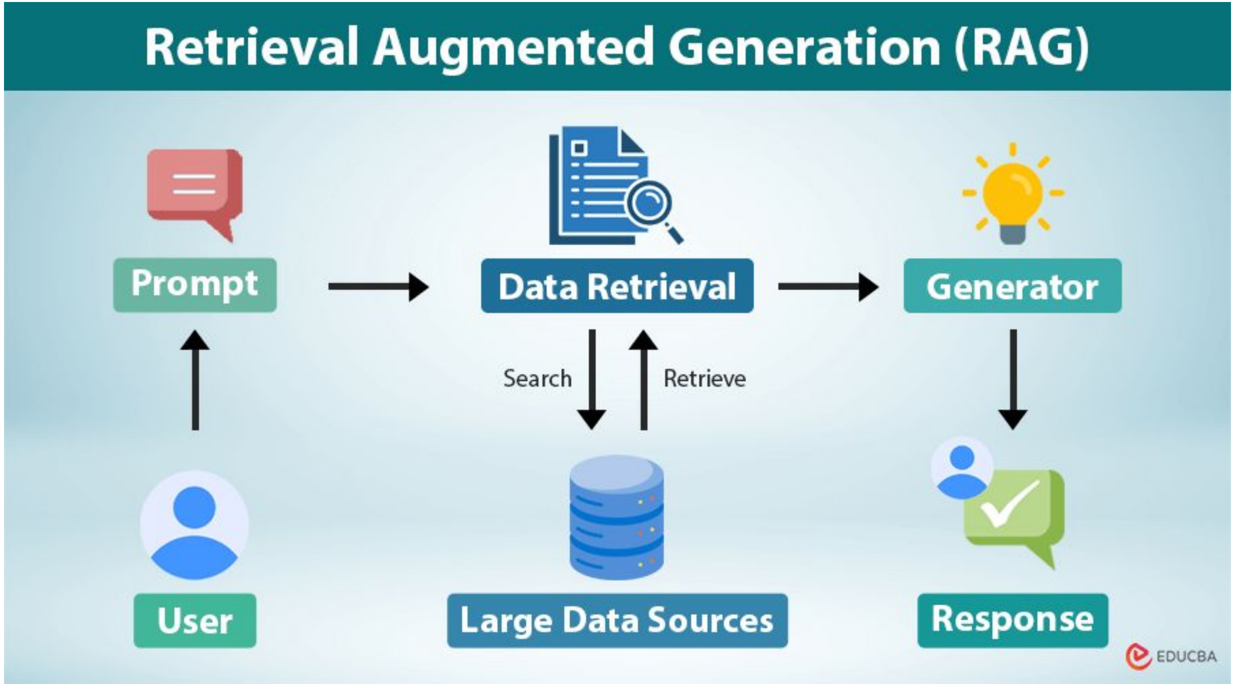
RAG Review

RAG Review

Once we’ve established some sort of **data-layer approach** to interacting with our relevant data, we can then implement an **application-layer** approach such as **retrieval augmented generation.**

Recall that this entails NO modification to the base LLM!

We simply “inform” LLMs by a**dding contextual information to the user prompt**. Think of this as giving our LLM an “open-textbook” exam where it is allowed to consult with resources before answering.



In **W10D1**, we created a RAG using a vector database. However the data source can also be Google, or some other specialized data source.

RAGs

Usage

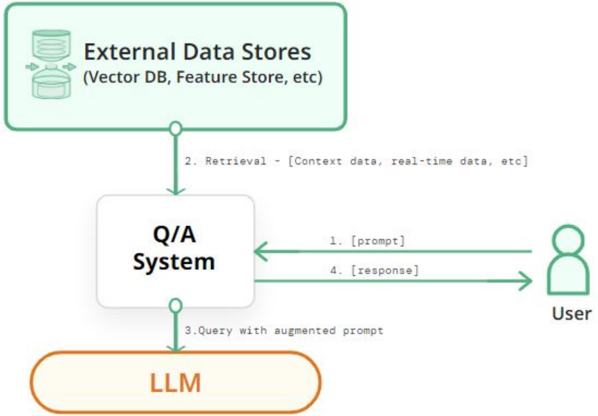
Combine LLMs with a retriever (often a vector DB) to inject external knowledge at runtime.

Pros

- Provides up-to data & **focused** answers
- **Reduces** hallucations
- Vector db acts as a run-time **persistent** memory

Cons

- Can still hallucinate answers if **retrieved data is too broad or irrelevant**
- Retrieval **slows down** answers


External Data Stores
(Vector DB, Feature Store, etc)
2. Retrieval - [Context data, real-time data, etc]
Q/A System
1. [prompt]
4. [response]
User
3.Query with augmented prompt
LLM

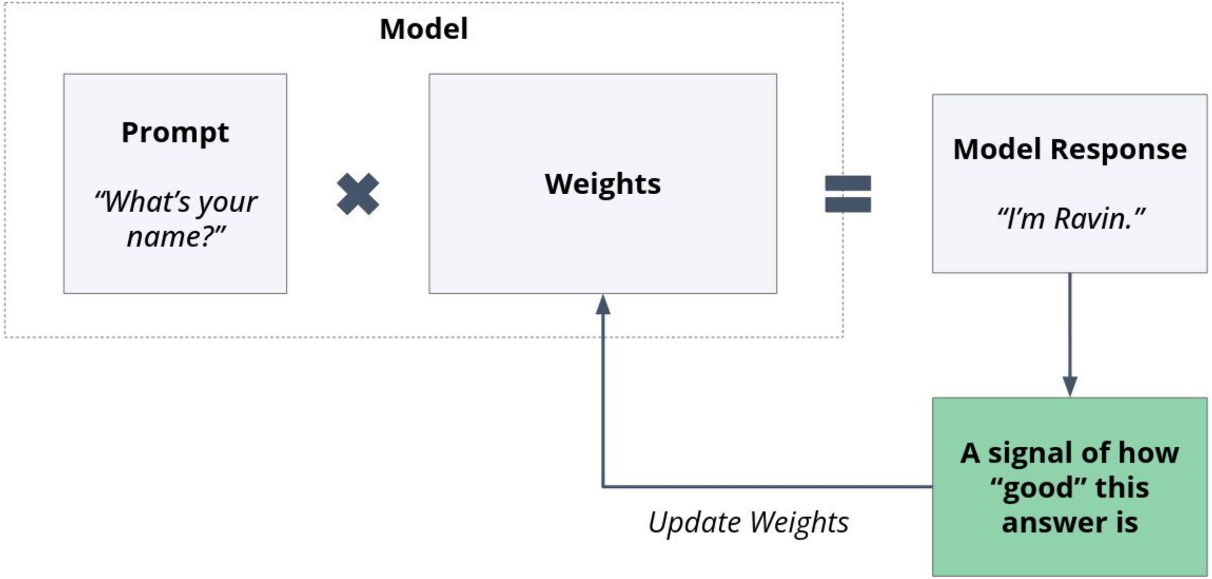
Fine-tuning Review

Fine-Tuned Models

Lastly, we also explored how we can modify LLM output by implementing the **model-layer application** of fine-tuning a model.

As we established, this is computationally expensive, however allows our model to be better-aligned with intended output format, tone, and knowledge.

We train LLMs just like any machine learning model by updating its' weights based on **the error that it makes**. Think of this as making our LLM study until it can successfully answer all questions in an exam.



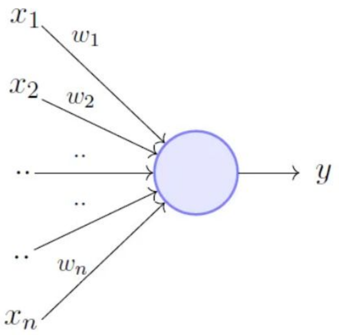
In **W10D2**, we created a fine-tuned model by providing a training set of intended outputs. We then measured the models' accuracy of responses on a testing set.

“Weights” clarification

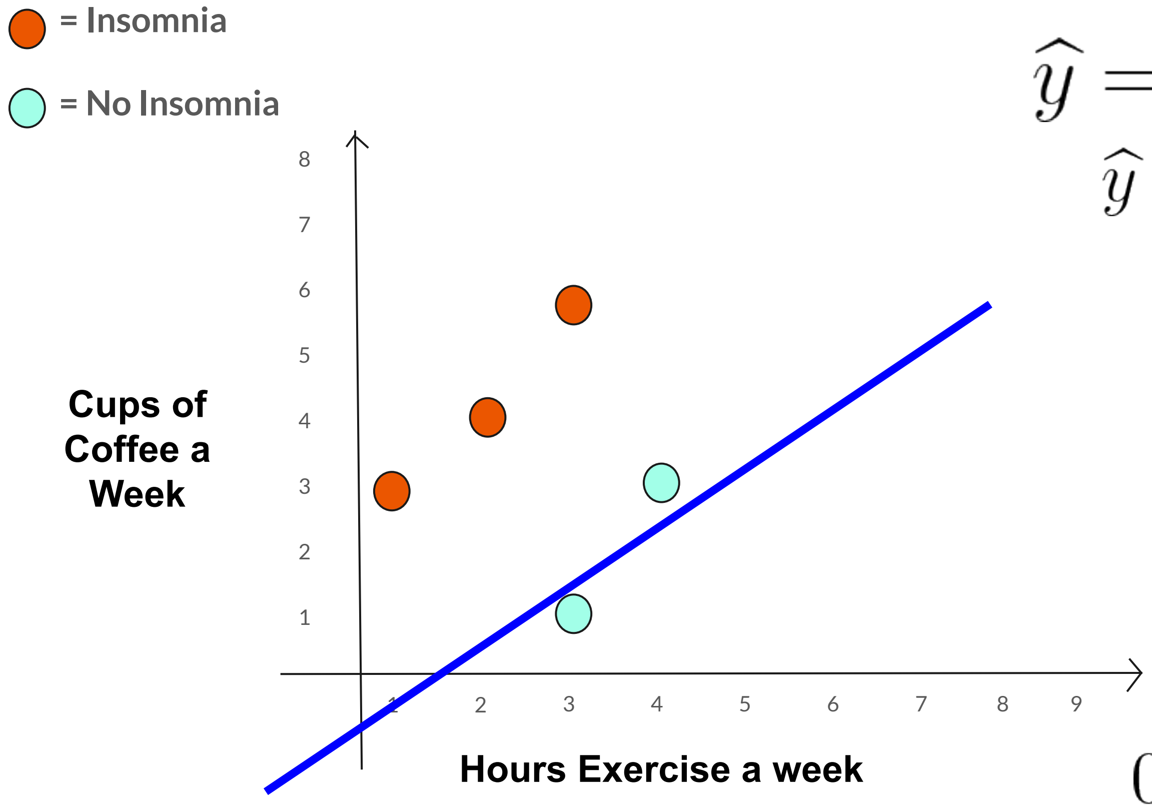
Remember that the concept of a “weight” or coefficient is by no means new. A “weight” is the numeric value that our model “learns” in order to predict some value.

- **Coefficients/Weights** (W)
- **Inputs** (X)

We express these as **vectors which we take the dot-product of.**



$$\widehat{y} = W^T X$$



$$\widehat{y} = w_0 + w_1 x_0 + w_2 x_1$$

$$\widehat{y} = 0.3 - 0.7x_1 + 1.3x_2$$

y1 = 0.3 - 0.7(1) + 1.3(3) = 3.5 → 1
y2 = 0.3 - 0.7(2) + 1.3(4) = 4.1 → 1
y3 = 0.3 - 0.7(3) + 1.3(6) = 6 → 1
y4 = 0.3 - 0.7(3) + 1.3(1) = -0.5 → 0
y5 = 0.3 - 0.7(4) + 1.3(3) = 1.4 → 1

Note this hyperplane is calculated by setting our equation greater than 0

$$0.3 - 0.7x_1 + 1.3x_2 \geq 0$$

We calculate our weights by **performing gradient descent & backpropagation**.

By solving the inequality b/w x1 & x2 we get the following hyperplane

Fine-Tuned Models

Usage

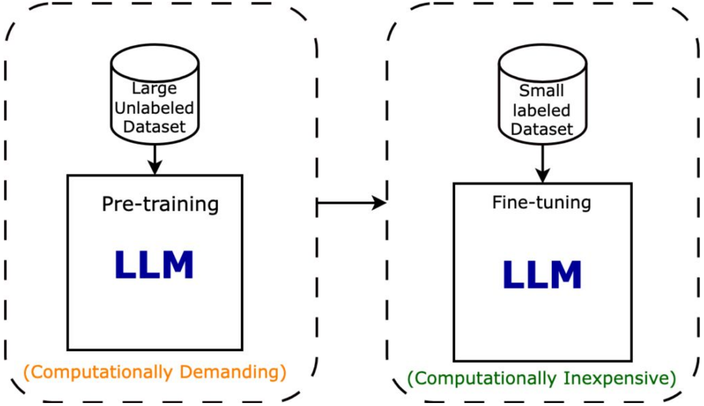
Adapt an LLM to a domain, style, or task by training on specialized datasets.

Pros

- No slow-down of retrieval, **fast inference**
- Answers become **focused** and domain-specific

Cons

- **Expensive** in time and $$$ to train
- Becomes **outdated** without retraining



Agents

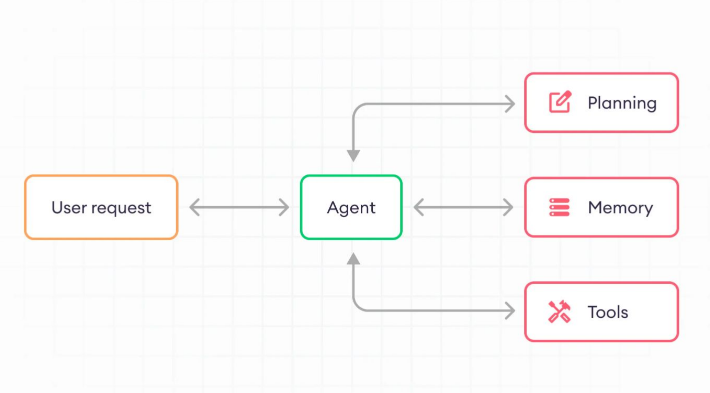
Agents

One issue with LLMs that we still haven't addressed is the inability to reason through multi-step problems.

One potential solution to this drawback are **agents.**

This boils down to creating an LLM agent that can:

- Call **external tools**
- Interact with other **LLM agents**
- Utilize RAGs to inform answers



Think of this as combining multiple LLM tools together to create a system of well-informed answers and self-management.

Agents

Usage

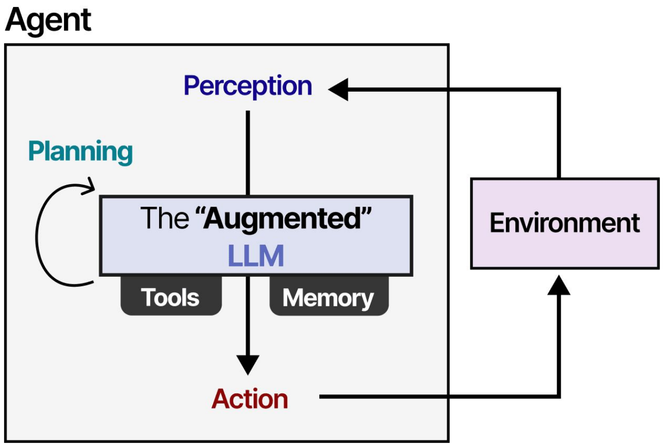
LLMs augmented with reasoning loops that can decide when and how to call tools, APIs, or other systems.

Pros

- Supports **complex reasoning** workflows
- When used in conjunction with RAGs/Fine-tuned models, all former pros of tools included

Cons

- **Expensive** in time to orchestrate
- More **points of failure** (vs just one)
- Requires robust **data management**


Agent
Perception
Planning
The "Augmented" LLM
Tools
Memory
Action
Environment