6.S978 AI ENGINEERING

LECTURE 3

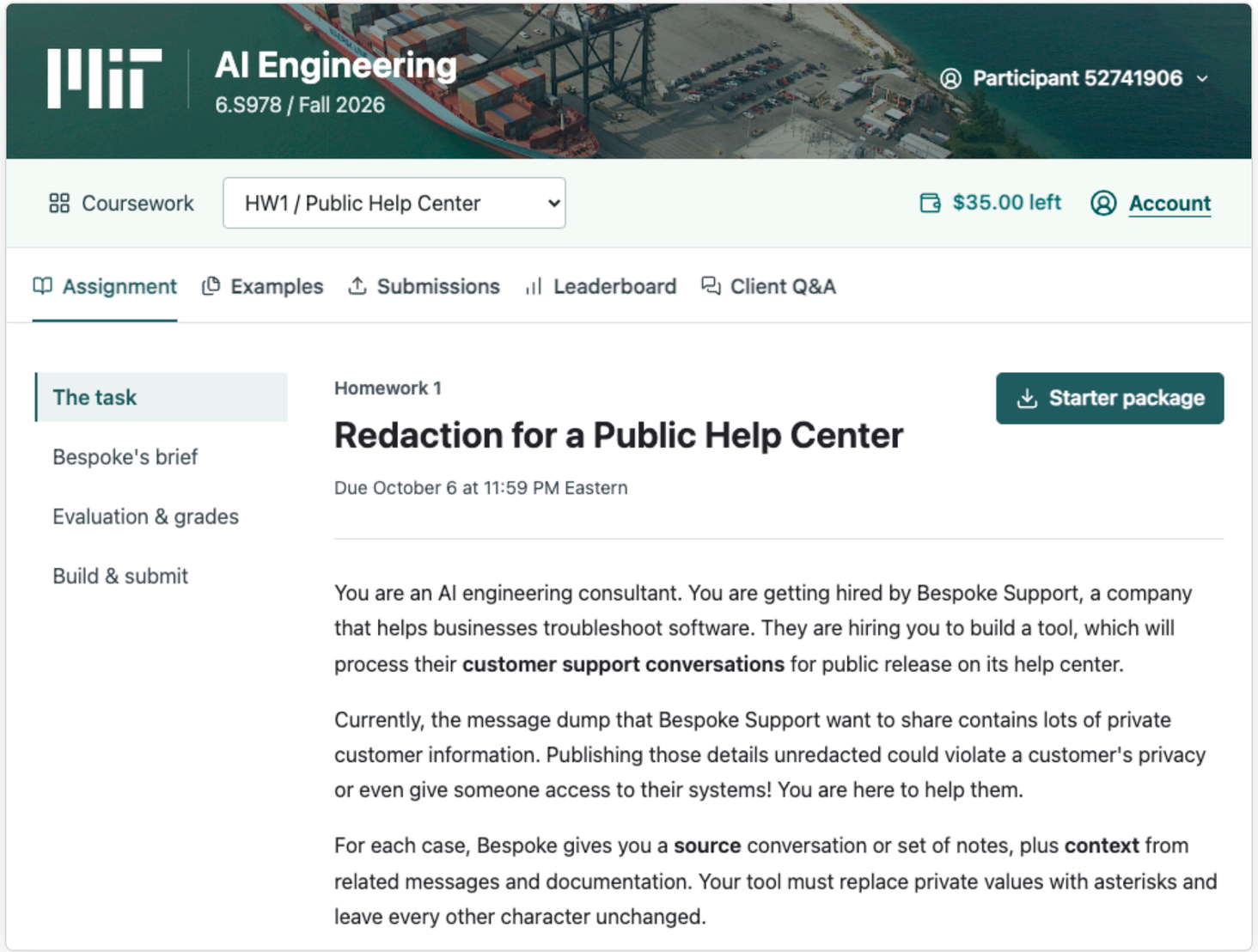
Declarative AI Programming

Omar Khattab

Engineering AI Systems and Agents

Quiz 1

test + discussion (45 minutes)



HW1

discussion
(10 minutes)

Requirements are **situated**. We elicit them from stakeholders, as natural language, code, and examples.

We asked: can we find a principled approach to engineering **intelligent systems** that are reliable, efficient, and maintainable?

So far, **reliability**. Today, **maintainability**.

Unfortunately, the **model changes** every few months.

How do we write an AI program that **keeps meeting its requirements**, while the **model keeps changing**?

Today

1 Maybe just wait for a **better model**?

2 How should we write **durable** AI programs?

SHORTER LECTURE DUE TO QUIZ AND HW DISCUSSION
TO BE CONTINUED IN LECTURE 4

Part 1

Maybe just wait for a better model?

Suppose it is **2020**. How would you answer this?!

> Did Red Flaherty ever umpire a game whose MVP made the Hall of Fame? **Yes.**

en.wikipedia.org/wiki/Red_Flaherty

Red Flaherty

From Wikipedia, the free encyclopedia

John Francis "Red" Flaherty (April 25, 1917 – April 1, 1999) was an American professional baseball umpire who worked in the American League from 1953 to 1973. … He umpired in four World Series (1955, 1958, 1965 and 1970), …

→

en.wikipedia.org/wiki/1965_World_Series

1965 World Series

From Wikipedia, the free encyclopedia

The series is best remembered for the heroics of Sandy Koufax who was named the World Series Most Valuable Player. …

→

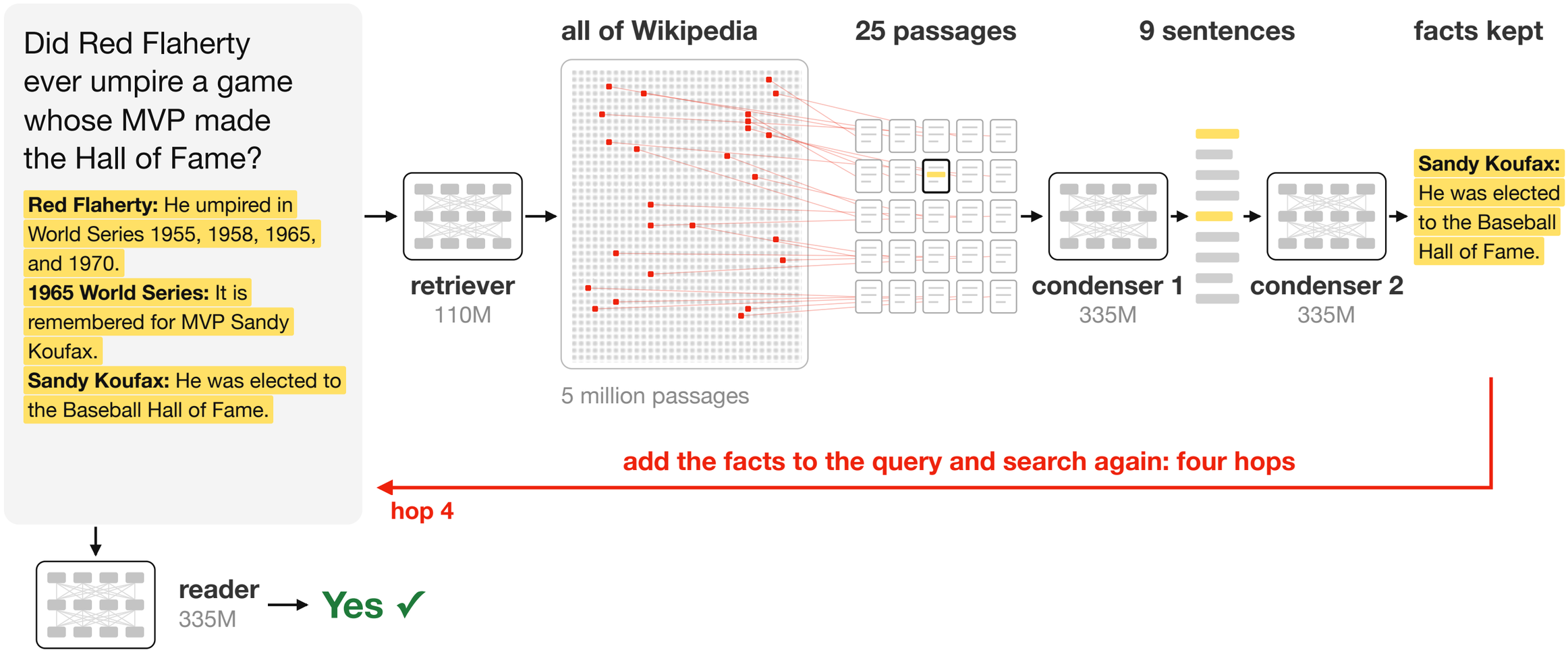
en.wikipedia.org/wiki/Sandy_Koufax

Sandy Koufax

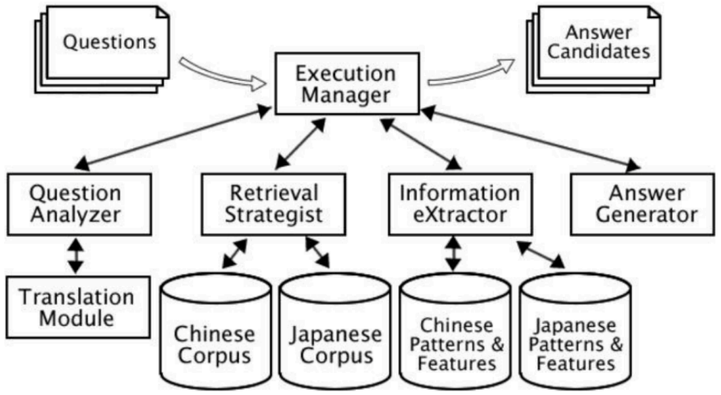
From Wikipedia, the free encyclopedia

Sanford Koufax (/ˈkoʊfæks/; né **Braun**; born December 30, 1935), nicknamed "**the Left Arm of God**", is an American former professional baseball player. … Retiring at age 30 due to chronic pain in his pitching elbow, Koufax was elected to the Baseball Hall of Fame in 1972 at age 36, the youngest player ever elected.

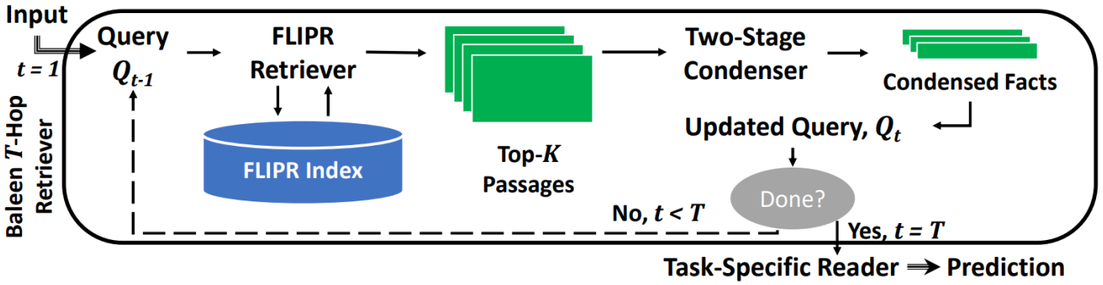
A claim from HoVer (Jiang et al., 2020), posed as a question, plus shortened Wikipedia screenshots.



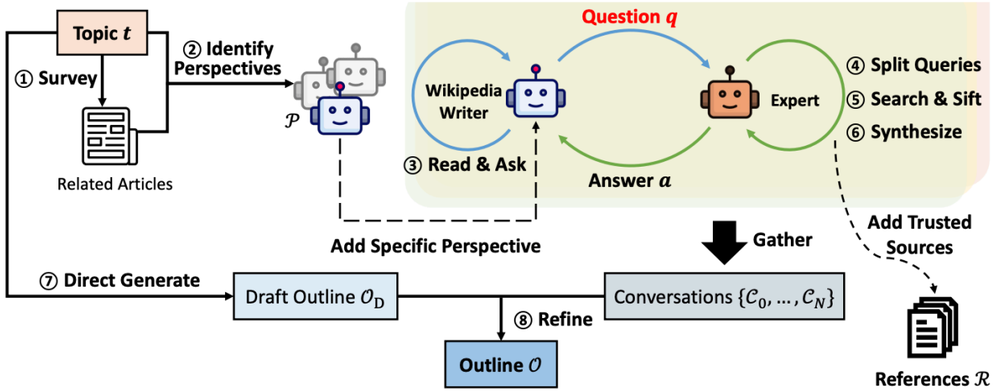
How would you train it?



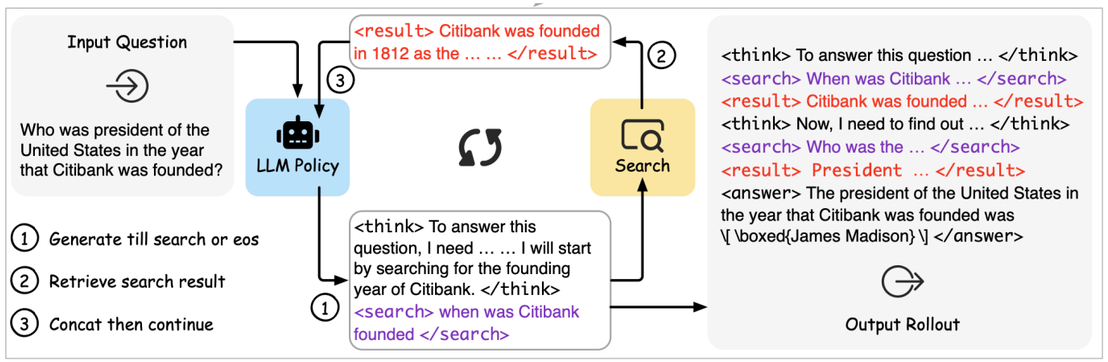
2006 Patterns and features, written by hand



2020 Networks, trained for each step



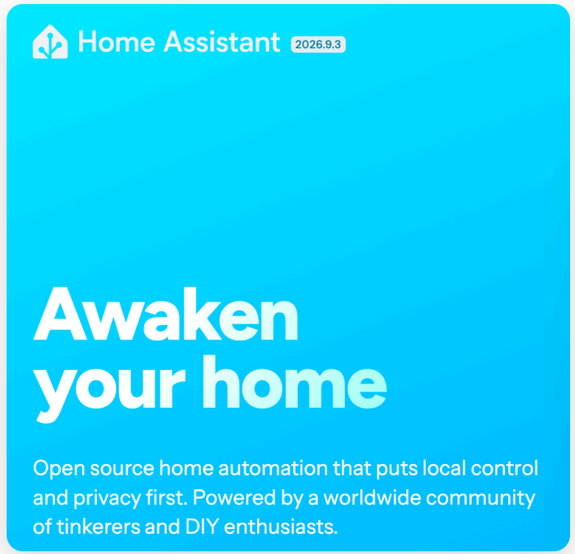
2024 LLMs, prompted for each step



2025 An LLM, trained to search

Is there a language in which their real differences are sparse? Maybe English??

Suppose it's **2024**. You're writing the prompt for **Home Assistant**.




```
You are a voice assistant for Home Assistant.
Answer in plain text.
Keep it simple and to the point.
```

JUNE 2024
"Pass prompt as system_instruction for Gemini 1.5 models"

JULY 2024
"Update LLM prompt to improve quality for local LLMs"

APRIL 2025
"Improve the live context tool prompt with additional instructions"

JUNE 2026
"Sort aliases in LLM prompts for stable prefix caching"

AUGUST 2026
"Fix default instructions to not suppress tool use"

What does the prompt look like now?

home-assistant.io, September 2026. home-assistant/core, helpers/llm.py: the instructions in June 2024, and commit messages since.

Today, September 2026 ↓

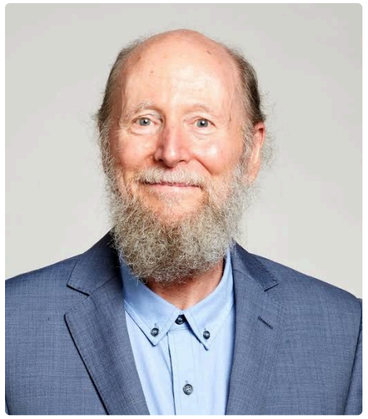
something it could infer — a nudge for today's model

```
You ARE equipped to answer questions about the current state of
the home using the `homeassistant__GetLiveContext` tool. This is a primary function.
Do not state you lack the functionality if the question requires live data.
If the user asks about device existence/type (e.g., "Do I have lights in the
bedroom?"): Answer from the static context below.
If the user asks about the CURRENT state, value, or mode (e.g., "Is the lock locked?",
"Is the fan on?", "What mode is the thermostat in?"):
    1. Recognize this requires live data.
    2. You MUST call `homeassistant__GetLiveContext`. This tool will provide
       the needed real-time information (like temperature, lock status, etc.).
    3. Use the tool's response** to answer the user accurately.
For general knowledge questions not about the home: Answer truthfully from
internal knowledge.
When controlling Home Assistant always call the intent tools.
Use HassTurnOn to lock and HassTurnOff to unlock a lock.
```

Promptese* entangles your system design with model-specific limitations.

***HW1?**

The Bitter Lesson



Rich Sutton

March 13, 2019

The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin. The ultimate reason for this is Moore's law, or rather its generalization of continued exponentially falling cost per unit of computation. Most AI research has been conducted as if the computation available to the agent were constant (in which case leveraging human knowledge would be one of the only ways to improve

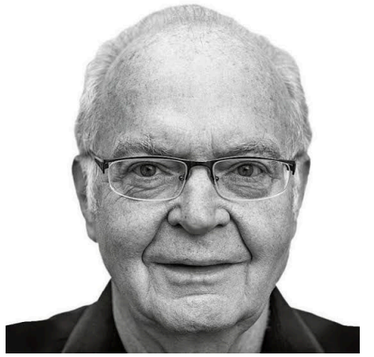
If someone tells you that leveraging domain knowledge is *that* bad, what is your answer to: what can AI *engineering* even be about?

"complicated methods get in the way of scaling"

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;  // evil floating point bit level hacking
    i  = 0x5f3759df - ( i >> 1 );  // what the @#$%?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );
    return y;
}
```

“Premature optimization is the root of all evil.”



Structured Programming with go to Statements

DONALD E. KNUTH

Stanford University, Stanford, California 94305

This is the bitter lesson for *software*, including AI software.

When does it count as premature optimization?

Hypothesis: it is premature when you hard-code at a lower level of abstraction than you can justify.

Part 2

How should we write durable AI programs?

1) **Symbolic Code** is a natural way to specify modular information flow and composition…

```python
def answer(question: str) -> bool:
    notes = []
    for hop in range(4):
        query = write_query(question, notes)
        passages = search(query, k=25)
        notes += keep_facts(question, passages)
    return decide(question, notes)
```

Unlike conventional raw prompts, code allows us to define…

1. independent, reusable **modules with reliable contracts**
2. **algorithms that compose** over these modules
3. decisions about how **information should (or should not) flow** between different modules

2) **Natural language** is essential for specification…

```
You are Prof. Einstein, an expert fact-checker.
Read the passages and think step by step.
Question: {question}
Passages:
[1] {title}: {text}
[2] {title}: {text}
Return ONLY a JSON list of sentences copied
from the passages. If none, return [].
Do NOT add commentary.
```

✗ **Wording to appease *this* LLM**

✗ **How inference strategy modifies your prompting**

✗ **How you format your data within instructions**

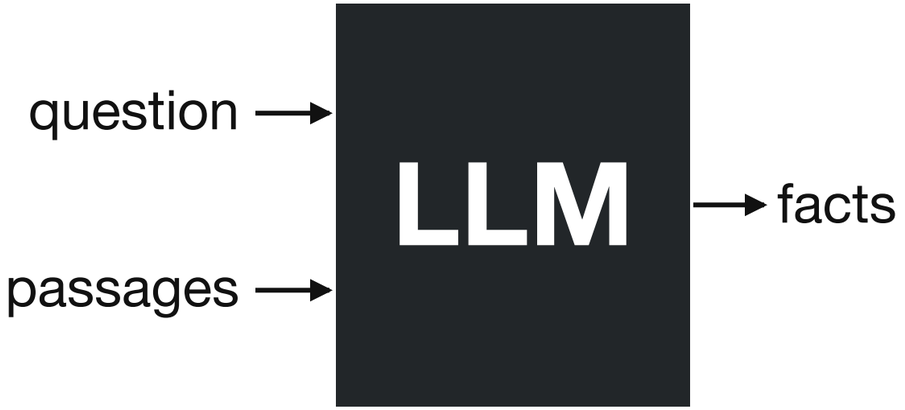
✗ **Output format and how you handle failures**

1 **Which facts are essential?**

2 **What are we assuming when we say so?**

3 **And how do we connect this to a larger program?**

Central abstraction: **typed** natural-language **signatures**




```
class KeepFacts(Signature):
    """Keep relevant facts."""

    question: str = InputField()
    passages: list[str] = InputField()

    facts: list[str] = OutputField(
        desc="copied from the passages")
```

Task

Inputs

Outputs

Functions with clear, typed contracts can localize ambiguity in a larger system and decouple what vs. how.

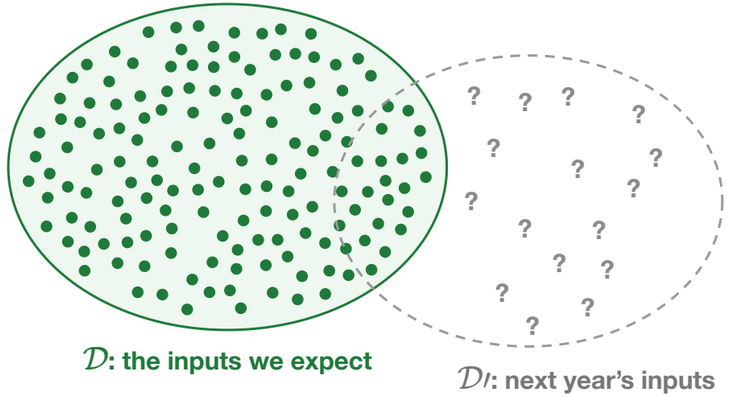
AI programs should just be traditional software that composes signatures into larger modular systems!

3) **Data (or feedback)** is a natural way to communicate a *distribution* of inputs, edge cases, rewards…

After the definitional natural-language priors and modular information flow are specified, edge cases are best specified **by example**.

Input	Metadata
Tricky question 1	The response should mention…
Tricky question 2	The response should cite…
Tricky question 3	The system should use tool…

Suppose 500 random questions all pass.



✗ **Assume the LLM will do the right thing.**

✓ **Assume next year's inputs still come from $\mathcal{D}$.**

Then $\Pr_{x \sim \mathcal{D}} \left[P(x) \models R \right]$ stays high.

A guarantee? But under an assumption about the inputs, not about the model. Lecture 5!

So is **modularity** good or bad?

```
answer = agent(question → answer: bool, tools=[search])
```

✗ Modularity that expresses **how** to run on today's model is **premature optimization**.

A good compiler rewrites it: it inlines, fuses, specializes.

✓ Modularity that expresses **what** you want is simply **system design**. Irreducible.

It lets you iterate, isolate changes, and assign credit.

Reliable systems actually *subtract* agency in the right places.

NEXT WEEK

Agent Design

How do we get the **most** out of a model,
without giving up reliability and maintainability?