# 140.800: How to AI (for Public Health)

## AI Agents and the ReAct Framework

Yiqun T. Chen
Email: yiqunc@jhu.edu
Schedule office hours via email

Departments of Biostatistics and Computer Science
Data Science & AI Initiative and Malone Center for Engineering in Health

# APIs as the Building Blocks of Action

**Q:** What can an LLM *not* do without tools?

# Capability 1: Accessing Real-Time Data

**Think:** What's the first thing you do when you need a current fact? You search for it.

A vanilla LLM's knowledge is frozen at its last training date. Agents overcome this by using tools to **query the live internet**.

- **Tool Used:** Web Search API (e.g., Google, Tavily).
- **Example Queries:**
  - "What's the weather like in Baltimore right now?"
  - "What was the final score of the Orioles game last night?"
  - "What is the current stock price of Google?"

# Capability 2: Grounding in Verifiable Sources

**The Goal:** How can we force an agent to "show its work" and fight hallucinations?

Instead of just the open web, agents can search specific, high-quality knowledge bases to **provide reliable answers with citations**.

- **Tool Used:** Database Connector (e.g., PubMed API, ArXiv, internal company wiki).
- **Example Task:**
  - **User:** "Summarize recent findings on metformin and cancer, with citations."
  - **Agent Action:** Query the PubMed API for relevant paper abstracts.
  - **Result:** A summary grounded in specific, verifiable scientific literature.

# Capability 3: Executing Code & Analyzing Data

**The Shift:** Moving from a linguistic model to a computational engine.

A vanilla LLM can *write* code, but an agent can **execute it in a secure environment** (like a Python interpreter). This unlocks all forms of quantitative reasoning.

- **Tool Used:** Code Interpreter / Jupyter Kernel.
- **Example Tasks:**
  - Solve a complex physics problem.
  - Analyze a user-uploaded CSV file to find trends.
  - Generate a data visualization plot.

# Capability 4: Taking Actions in the World

**The Goal:** Turning natural language into meaningful, real-world actions.

Agents can connect to thousands of third-party APIs to **operate other software and services** on the user's behalf.

- **Tools Used:** Google Calendar API, Gmail API, Zapier, etc.
- **Example Actions:**
  - "Find me a flight to San Francisco next Tuesday and put a hold on it."
  - "Draft an email to my team about the new project deadline."
  - "Add a dentist appointment to my calendar for Friday at 3 PM."

# More on Grounding in Verifiable Sources

**Think:** Why is an unsourced answer from an LLM potentially dangerous?

- **Fact Fabrication (Hallucination):**
  - Models confidently state incorrect "facts."
  - They may invent plausible but fake URLs, book titles, or scientific citations.
- **Lack of Attribution:**
  - Answers are generated from a "black box" with no sources.
  - It is impossible to verify *where* the information came from.
- **Stale Knowledge:** The world moves on, but the LLM's data is frozen in time.

# Solution: Retrieval-Augmented Generation (RAG)

**The Core Idea:** What if we gave the LLM an "open-book" exam?

Instead of relying solely on its internal (and sometimes faulty) memory, we first **retrieve** relevant information from a trusted, external knowledge source.

RAG combines the best of two worlds:

- **Fast Information Retrieval**: To find relevant, up-to-date facts from a specific knowledge base (e.g., recent papers, company docs).
- **Fluent Language Models**: To synthesize the retrieved facts into a coherent, human-readable answer.

# How RAG Works: Retrieve → Generate

**Trace the data:** Where does the "grounding" happen in RAG?

# Building a RAG System:

**The Goal:** How do we move from a simple mechanism to an intelligent process?

An advanced RAG system is not a fixed pipeline. It's a dynamic process that must intelligently answer three core questions at every step:

1. **What should I retrieve?**
2. **How is knowledge represented?**
3. **How do I synthesize the results?**

# Step 1: Deciding What to Retrieve

**Think:** Do you use the same search strategy for every question you have?

# Step 1: Deciding What to Retrieve

**Think:** Do you use the same search strategy for every question you have? An intelligent agent must first understand the user's intent and then select the best source and query method.

- **Adaptive Retrieval (When to search):**
  - The agent first decides if retrieval is even necessary. For common knowledge ("What is the capital of France?"), it can answer from memory.
- **Query Transformation (How to ask):**
  - The agent rewrites a user's vague query into several precise, targeted questions optimized for the knowledge base.
- **Source Selection (Where to look):**
  - The agent chooses the best knowledge source: searching the web for current events, a vector database for internal documents, or a knowledge graph for factual relationships.

# Strategy 2: How to Represent Knowledge

**The Challenge:** How do you prepare documents so a machine can find the relevant information?

# Strategy 2: How to Represent Knowledge

The core idea is to convert text into numerical vectors and find the ones that are closest to the user's query in a high-dimensional space. This involves three steps:

1. **Chunk:** Split original documents into small, manageable text passages ($c_1, c_2, \ldots, c_n$).

2. **Embed:** Convert each text chunk $c$ into an embedding vector $\mathbf{c}$.

3. **Search:** The user's query $q$ is also embedded into a vector $\mathbf{q}$. The system then calculates the similarity between the query vector and all chunk vectors, often using cosine similarity. We then return the top-N chunks.

# Strategy 2: How to Represent Knowledge

The quality of retrieval depends entirely on how well the underlying knowledge is structured and indexed.

- **Intelligent Chunking:**
  - Instead of splitting documents into arbitrary fixed-size pieces, **semantic chunking** divides text based on topical shifts. This ensures each chunk contains a complete thought.
- **Multi-Representation Indexing:**
  - A document is indexed in multiple ways. We can create and embed:
  - **Summaries** of long passages.
  - **Hypothetical questions** that a chunk answers.
  - This gives the retriever more ways to match a query to the right context.

# Strategy 3: How to Synthesize Information

**The Final Step:** What should the LLM do with the retrieved documents?

# Strategy 3: How to Synthesize Information

**The Final Step:** What should the LLM do with a dozen retrieved documents?

- **Re-ranking for Relevance:**
  - The initial retrieval might return 20+ documents. A fast, secondary model skims these results to push the most relevant ones to the top.

- **Iterative Refinement & Self-Correction:**
  - The LLM doesn't trust the first retrieval pass. It checks if the context is sufficient. If not, it can autonomously trigger a new search with a refined query.

- **Handling Contradictions:**
  - If retrieved sources disagree, an advanced LLM can highlight the conflict or even try to determine which source is more credible.

# Another Motivation: the Brittleness of LLM Arithmetic

LLMs are next-token predictors, not logical calculators. They are designed to recognize and replicate linguistic patterns, not to execute mathematical rules. This works for common facts but fails for complex calculations.

**Easy Problem (Memorized Pattern)**

- `User: 2 * 8 = ?`
- `LLM: 16`
- The answer is **recalled** from patterns in its training data, not computed.

**Hard Problem (Requires Logic)**

- `User: 12345 * 54321 = ?`
- `LLM: 670,592,745` (Wrong!)
- The LLM is just guessing tokens, not carrying digits.
- (Correct answer: 670,481,265)

If you can't *be* the calculator...

...why not just *use* one?

**This insight is the foundation of modern AI agents.**

# The Solution: From Doer to Delegator

**Intuition:** The LLM's real skill is translating language into code. The new approach is to teach the LLM to recognize a math problem and **delegate it to a tool that is perfect for the job**: a calculator like Python or an R interpreter.

# The Solution: From Doer to Delegator

**Intuition:** The LLM's real skill is translating language into code. The new approach is to teach the LLM to recognize a math problem and **delegate it to a tool that is perfect for the job**: a calculator like Python or an R interpreter.

1. **Thought:** The user is asking for a precise calculation. My internal arithmetic is unreliable. I will use my Python interpreter tool.
2. **Action (Code Generation):** The LLM writes code to solve the problem.
   ```
   print(12345 * 54321)
   ```
3. **Observation (Tool Output):** The interpreter executes the code and returns the correct result.
   ```
   670481265
   ```
4. **Final Answer:** The LLM uses the tool's output to give the user a reliable answer.

# The Tool-Use Revolution: Beyond the Calculator

**The Insight:** The code interpreter is just one tool. What if the LLM could use any program?

By treating any software with an API as a potential tool, the LLM evolves from a text generator into a universal controller for digital tasks.

**Example Toolkit:**

- **Web Browsers & Search Engines:**
  - To answer questions about current events or find the latest research.
- **Databases (e.g., SQL):**
  - To query company sales data or retrieve user information from a structured database.
- **Third-Party APIs:**
  - To book a flight, reserve a table at a restaurant, or add an event to a calendar.

We have the tools. But how does the LLM decide...

...what to do next?

**How do you solve a complex problem that requires multiple steps and multiple tools?**

# Philosophy 1: Plan-then-Execute

First, create a complete plan, then follow it exactly.

# Philosophy 1: Plan-then-Execute

First, create a complete plan, then follow it exactly.

In this model, the LLM first thinks through the entire problem and **generates a complete, multi-step plan from start to finish**. It then executes each step of that plan sequentially without deviation.

- **Analogy:** Writing a detailed, step-by-step recipe before you start cooking.
- **Key Characteristic:** Reasoning is done entirely upfront. The execution phase is "blind" to real-time events.
- **Primary Risk:** This approach is brittle. If an early step fails or produces an unexpected result, the entire pre-made plan can be derailed.

# Philosophy 2: Interleaved Reasoning & Acting

**A Flexible Alternative:** Think a little, act a little, and repeat.

# Philosophy 2: Interleaved Reasoning & Acting

**A Flexible Alternative:** Think a little, act a little, and repeat.

This alternative avoids rigid, long-term plans. Instead, the LLM interleaves its reasoning process with actions, adapting its strategy based on real-time feedback from its tools.

- **Analogy:** Navigating a new city with a map, checking your position and adjusting your route after every turn.
- **Key Characteristic:** Reasoning and acting are tightly coupled in a dynamic loop.
- **Primary Advantage:** This method is highly adaptive and can handle unexpected outcomes and change its plan on the fly.

# A Framework for Action: ReAct

**The Key Insight:** Why not build a system that explicitly combines reasoning and acting?

The **ReAct** framework formalizes this second, more flexible philosophy. The name itself reveals the core mechanic of this powerful paradigm:

# **Reason** + **Act**

- It's a general method for getting LLMs to solve complex tasks by interleaving **internal reasoning** (verbal thoughts) with **external actions** (tool use).
- This creates a dynamic and reliable problem-solving loop: **Thought** $\rightarrow$ **Action** $\rightarrow$ **Observation**

# ReAct in One Slide

# Beyond the Loop: Key Implementation Challenges

**The Question:** What makes building a reliable ReAct agent so difficult in practice?

Simply creating the loop is not enough. Several critical components determine whether the agent succeeds or fails.

**Key design considerations include:**

1. **Prompt Craftsmanship:** How do you instruct the LLM to think and act effectively?
2. **Robust Output Parsing:** How do you reliably interpret the LLM's output?
3. **Error Handling:** How can the LLM recover when a tool fails?
4. **Context & Memory Management:** How does the LLM handle long tasks?

# Challenge 1: Prompt Craftsmanship

**The Goal:** How do you teach an LLM the rules of the game?

The main system prompt is often the most critical part of a ReAct agent. It acts as an "operating system" that defines the agent's persona, **its available tools**, and the format for its reasoning process.

**Key Components of a ReAct Prompt:**

- **Persona:** "You are a helpful research assistant..."
- **Tool Definitions:** A list of available tools and their descriptions.
- **Formatting Instructions (The Contract):** A strict specification for how to format Thoughts and Actions.
- Example: Your response must be a JSON with "thought" and "action" keys. The action must be TOOL_NAME[ARGUMENT].

# Challenge 2: Robust Output Parsing

**The Problem:** What happens when the LLM doesn't follow instructions perfectly?

The LLM's output is just a string of text. Your code must reliably parse this string to extract a valid tool name and its arguments, even when the output is malformed.

# Challenge 2: Robust Output Parsing

**The Problem:** What happens when the LLM doesn't follow instructions perfectly?

The LLM's output is just a string of text. Your code must reliably parse this string to extract a valid tool name and its arguments, even when the output is malformed.

**Common Failure Modes to Handle:**

- **Invalid Format:** The LLM fails to generate valid JSON or the correct 'TOOL_NAME[ARGUMENT]' structure.
- **Hallucinated Tools:** The LLM calls a tool that doesn't exist (e.g., '`google_scholar_search[...]`').
- **Incorrect Arguments:** The LLM provides an argument with the wrong data type (e.g., a string instead of a number).

A robust parser should catch these errors and feed them back to the LLM.

# Challenge 3: Error Handling & Self-Correction

**The Question:** How does an agent learn from its mistakes?

# Challenge 3: Error Handling & Self-Correction

**The Question:** How does an agent learn from its mistakes?

**The Error Feedback Loop in Action:**

1. **Action:** LLM calls a tool, e.g., ‘`run_code[print(x/0)]`‘.
2. **Execution Fails:** The code interpreter throws a ‘ZeroDivisionError‘.
3. **Observation:** This error message is passed back to the LLM as the observation.
   `Observation: ZeroDivisionError: division by zero.`
4. **Next Thought:** The LLM sees the error and can now reason about it.

# Challenge 4: Context & Memory Management

**The Challenge:** How can an agent remember the beginning of a long task?

The ‘Thought-Action-Observation‘ history grows with every turn, and it will become really long for complex tasks.

# Challenge 4: Context & Memory Management

**The Challenge:** How can an agent remember the beginning of a long task?

The ‘Thought-Action-Observation‘ history grows with every turn, and it will become really long for complex tasks.

**Common Strategies for Memory:**

- **Windowed History:** Only keep the last ‘k‘ turns in the context. (Simple, but can forget important early information).
- **Summarization:** Use another LLM call to periodically summarize the history so far. (More effective, but adds latency and cost).
- **Vector-Based Retrieval:** Store each step in a vector database and retrieve the most relevant past steps for the current context. (Most complex, but very powerful).

# ReAct: Putting It All Together

## So, what are the key takeaways?

**In Summary:**

- The **'Thought-Action-Observation' loop** is the core engine that allows an LLM to reason, act, and adapt to new information.

- A reliable agent, however, requires a full support system around this loop, including:
  - A well-crafted **prompt** to define the rules.
  - A robust **parser** to interpret the LLM's output.
  - A strategy for **error handling** and **memory management**.

Now, let's see how these concepts map to a practical example.

# A Vision: The AI Research Assistant

**What if every scientist had an AI partner for their research?**

We can envision an agent participating in the entire scientific process, tackling tasks of increasing complexity, creativity, and real-world understanding.

**A Spectrum of Agentic Scientific Workflows:**

→ **Literature Synthesis** (Today's Capability)
Finding, summarizing, and connecting existing knowledge.

→ **Hypothesis Generation** (Emerging Capability)
Identifying gaps and proposing novel, testable ideas.

→ **Experimental Design** (The Frontier)
Designing valid experiments, from power calculations to lab protocols.

# Workflow 1: Automated Literature Synthesis

## Why is this task a perfect fit for today's ReAct agents?

This workflow is a multi-step information retrieval and synthesis problem, which maps directly to the capabilities of a tool-using LLM. It's the scientific application of the patterns we've already discussed.

**The Agent's Process:**

1. Use a ‘pubmed_search‘ tool to find relevant papers.
2. Use a ‘read_pdf‘ tool to extract the text from each one.
3. Use internal reasoning (generation) to summarize each paper.
4. Use internal reasoning again to synthesize a final report connecting the claims from all sources.

# Workflow 2: Hypothesis Generation

## What makes this harder than just summarizing papers?

Hypothesis generation requires the agent to go beyond retrieving known facts and start inferring unseen connections. This is a leap from information retrieval to knowledge creation.

**Key Challenges for the Agent:**

# Workflow 2: Hypothesis Generation

## What makes this harder than just summarizing papers?

Hypothesis generation requires the agent to go beyond retrieving known facts and start inferring unseen connections. This is a leap from information retrieval to knowledge creation.

**Key Challenges for the Agent:**

- **Identifying Gaps:** It must first build a knowledge map from the literature and then identify what is *missing*.
- **Creative Inference:** It needs to propose novel links between disparate concepts (e.g., connecting a biological pathway from one paper to a disease mechanism in another).
- **Requires New Tools:** The agent might need tools to traverse knowledge graphs (e.g., gene-protein interaction networks) to find plausible connections.

# The Frontier: Automated Experimental Design

## What's the ultimate challenge for an AI research assistant?

Designing a valid experiment is the bridge from the digital world of information to the physical world of action. This requires a much deeper level of understanding.

**This Task Requires the Agent to Understand:**

- **Causality:** The difference between correlation and causation, and how to design controls to isolate variables.
- **Resource Constraints:** The real-world limitations of budget, time, and equipment.
- **Physical Procedures:** Writing a step-by-step lab protocol that a human (or a robot) can actually execute.

# The Engine Powering the Vision

## How does this simple loop relate to these grand scientific challenges?

At its core, every workflow we've discussed is powered by this fundamental ReAct loop. The complexity lies in the tools provided and the sophistication of the LLM's reasoning.

```
def run_agent(query, tools):
    template = """..."""
    scratchpad, history = "", []

    while not done():
        prompt = template.format(...)
        response = llm.generate(prompt)
        thought, action = parse(response)
        scratchpad += f"Thought: {thought}\n"
        obs = execute(action, tools)
        scratchpad += f"Observation: {obs}\n"

    return final_answer
```

**Challenge:** What tools and reasoning capabilities would this loop need for true hypothesis generation?