# Building Deep Agents

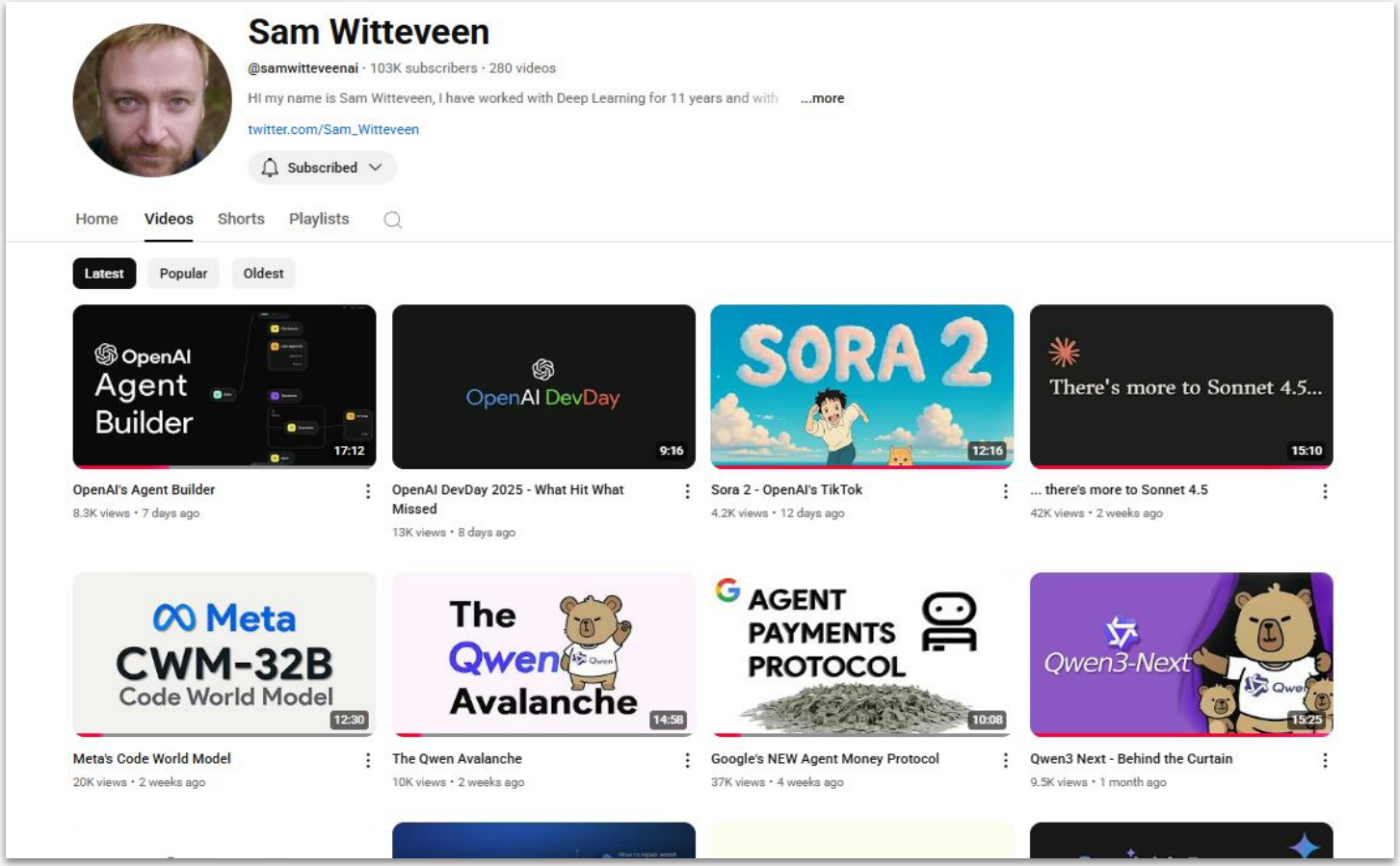
Sam Witteveen - Co Founder & CEO Red Dragon AI
Google Developer Expert Machine Learning & Deep Learning
sam@reddragon.ai
@sam_witteveen

# About me

- Google Developer Expert for Machine Learning and Deep Learning - 2017
- Deep Learning - Language & Dialogue, LLMs
- Multiple Startups - B2B and B2C
- Google for Startups Accelerator Mentor - Asia & North America
- Gemini Tester
- Based in Singapore
- Red Dragon AI

RED DRAGON AI

# Youtube - https://goo.gle/SamWitteveen



# Agents 1.0


2025
THE YEAR OF AGENTS

# What is an Agent?

An AI system that operates in a simple, stateless loop.

## How it works:

1. **User Prompt**: "What's the weather in Bedok?"
2. **LLM Reasons**: "I need a weather tool."
3. **Tool Call**: `get_weather("Bedok Singapore")`
4. **Observation**: Tool returns data.
5. **LLM Answers**: Generates a response.
6. **Repeat**.

The agent's entire "brain" and memory exist only within the LLM's context window.

# The Limits of Basic Agents

## Why Simple Agents Fall Short on Complex Tasks

They are great for 5-15 step tasks, but fail on 500-step projects due to:

- **Context Overflow:** The conversation history fills up with tool outputs (HTML, raw data), pushing original instructions and goals out of context.
- **Loss of Goal:** Amidst the noise of intermediate steps, the agent forgets the original, high-level objective.
- **No Recovery Mechanism:** If it goes down a wrong path, it rarely has the ability to stop, backtrack, and try a new approach.

# Low Level

# The LLM Equation

output = f(**input**)


**P( next_token | prompt)**

# The Agents Equation

output = f(**context** + **instruction/query**)

where….

**context** = (memory + exemplars + current status

+tool returns + Human feedback)

# Agents - At a low level

output = next step (Context)

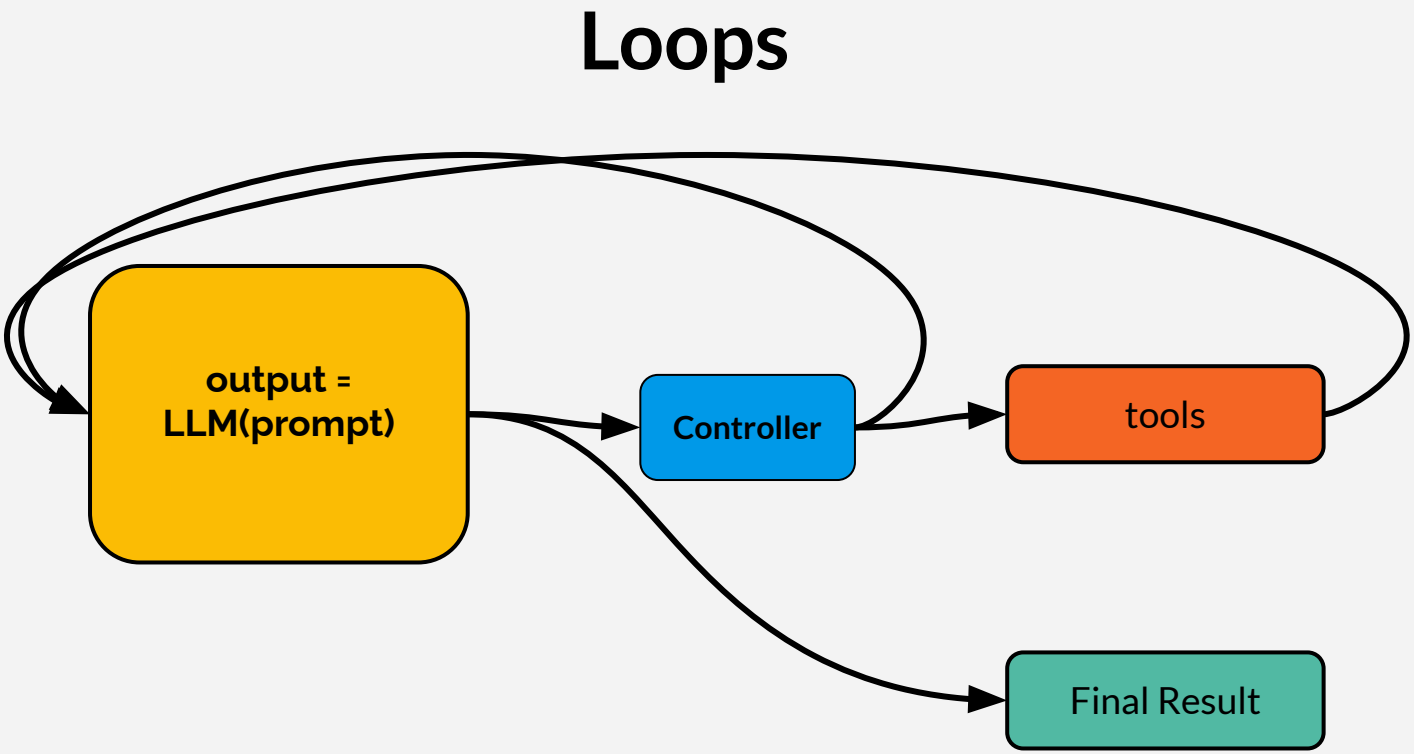
Eg,

output ≅ call a tool

output ≅ reasoning step

output ≅ final answer

# (Simple) Agents - At a low level



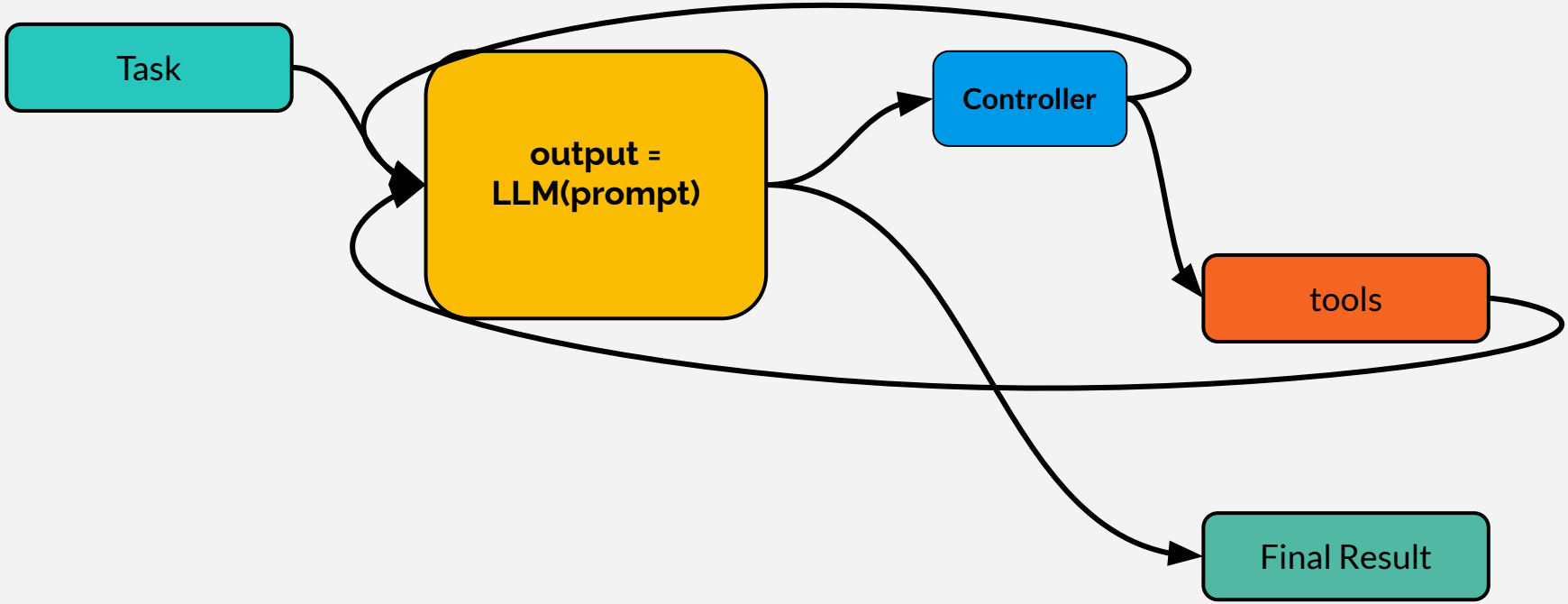
# The Myth of the Better Model

# Agents 1.5

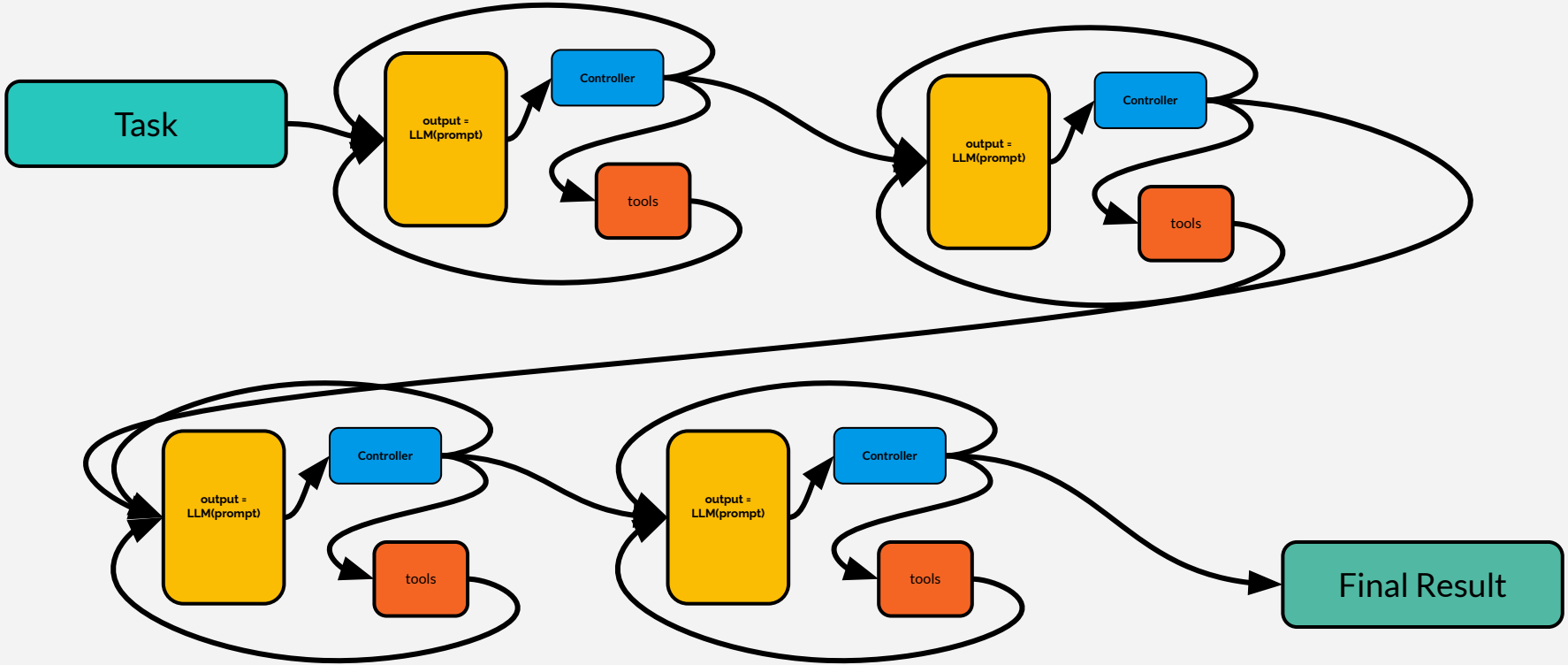
# Flows vs Agency

- Do you want something that has a predetermined flow?
- Do you want something with full autonomy?
- What is the cost of it going wrong?
- Do you have a checker/verifier to keep the agent on track?

# Full Agency

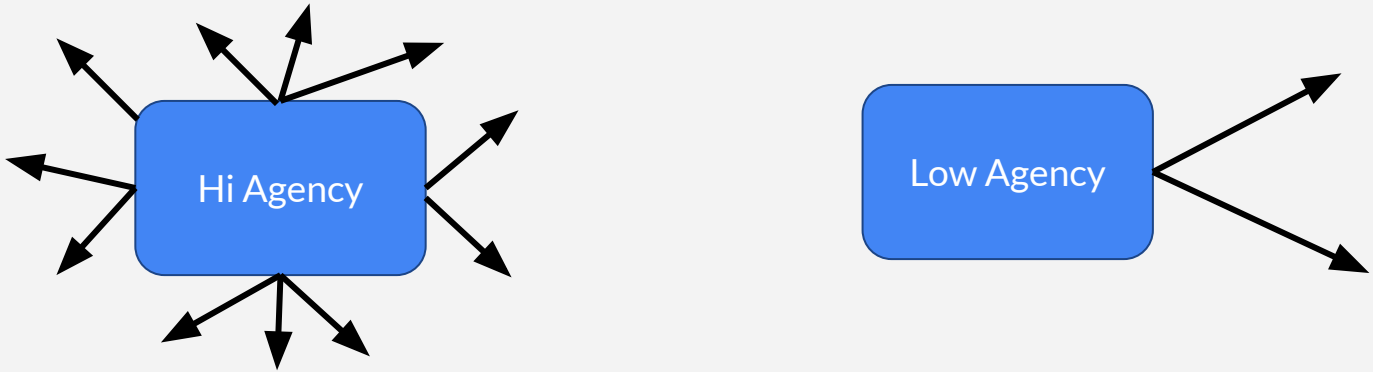


# Flow Engineering



# The Level of Agency & Autonomy

- More Agency means more the Agent can do (in theory)
- … but More Agency means more can go wrong



# On Rails

- The more you limit the choices the less things can go wrong

# What is a Deep Agent? (SuperAgents)

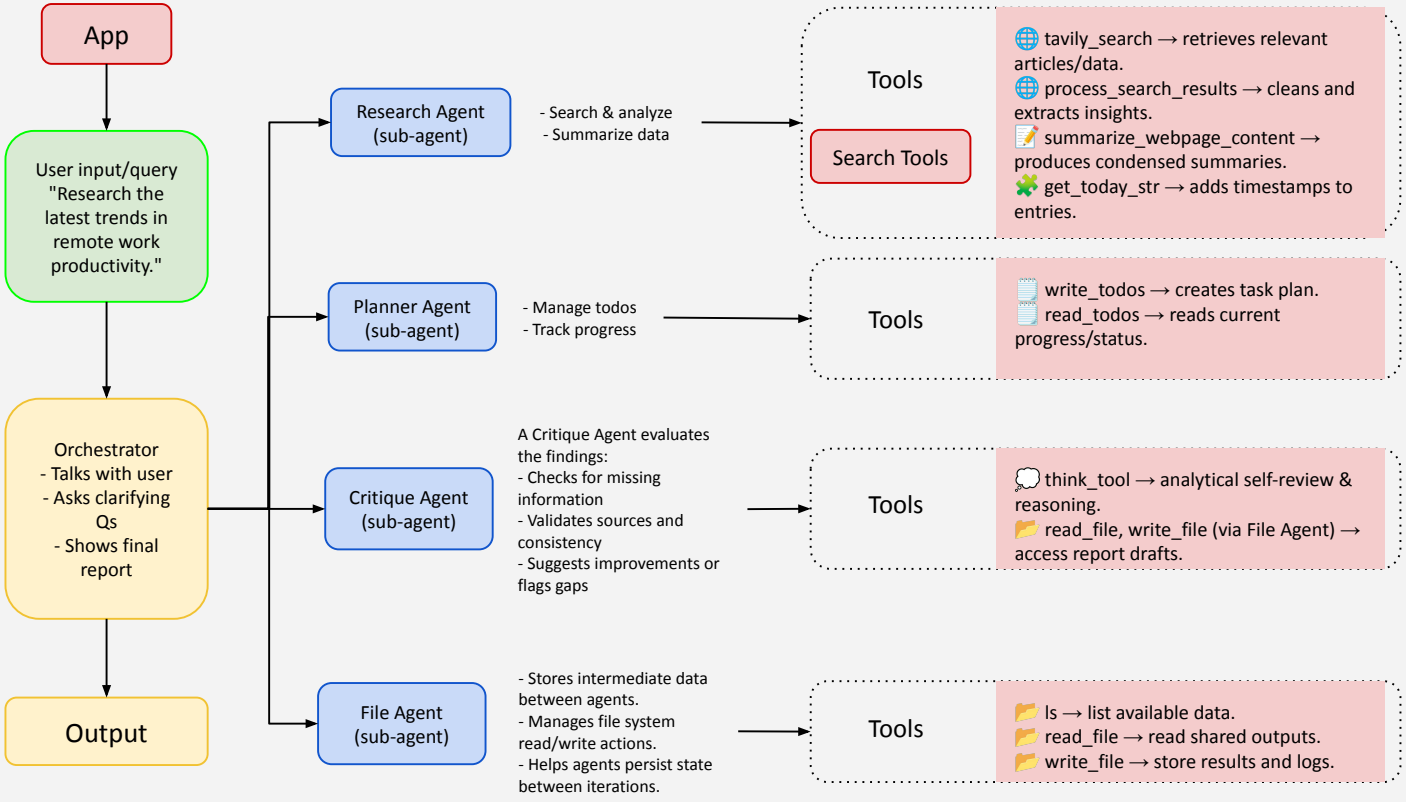
## The Evolution: "Deep" Agents (Agent 2.0)

An architectural shift from reactive loops to proactive systems capable of handling complex, long-horizon tasks + well controlled flow engineering.

Key Characteristics:

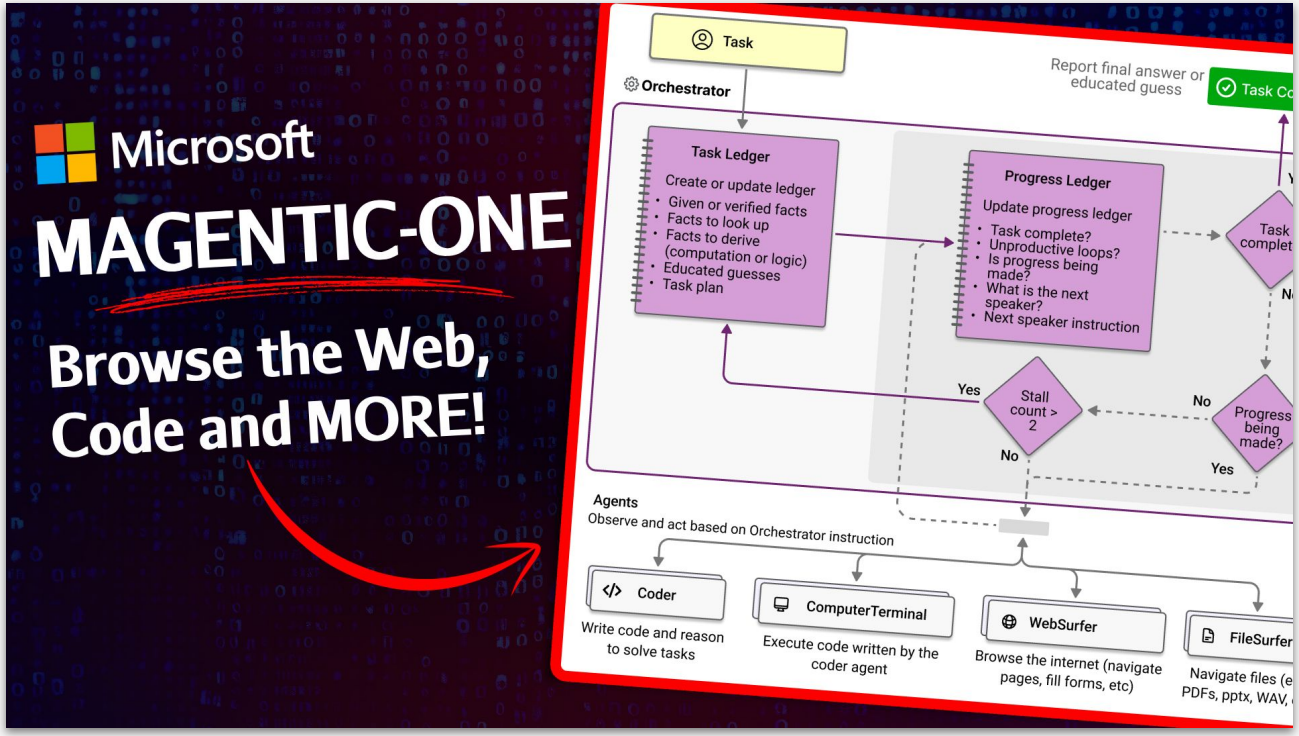
- **Decouples planning from execution**.
- **Manages a persistent state/memory external** to the context window.
- **Delegates work** to specialized sub-agents.
- **Solves problems with longer time runs**, not just seconds.

# DeepAgents


App
User input/query "Research the latest trends in remote work productivity."
Orchestrator
- Talks with user
- Asks clarifying Qs
- Shows final report
Output
Research Agent (sub-agent)
- Search & analyze
- Summarize data
Planner Agent (sub-agent)
- Manage todos
- Track progress
Critique Agent (sub-agent)
A Critique Agent evaluates the findings:
- Checks for missing information
- Validates sources and consistency
- Suggests improvements or flags gaps
File Agent (sub-agent)
- Stores intermediate data between agents.
- Manages file system read/write actions.
- Helps agents persist state between iterations.
Tools
Search Tools
🌐 tavily_search → retrieves relevant articles/data.
🌐 process_search_results → cleans and extracts insights.
📝 summarize_webpage_content → produces condensed summaries.
🧩 get_today_str → adds timestamps to entries.
Tools
🗒️ write_todos → creates task plan.
🗒️ read_todos → reads current progress/status.
Tools
💭 think_tool → analytical self-review & reasoning.
📂 read_file, write_file (via File Agent) → access report drafts.
Tools
📂 ls → list available data.
📂 read_file → read shared outputs.
📂 write_file → store results and logs.

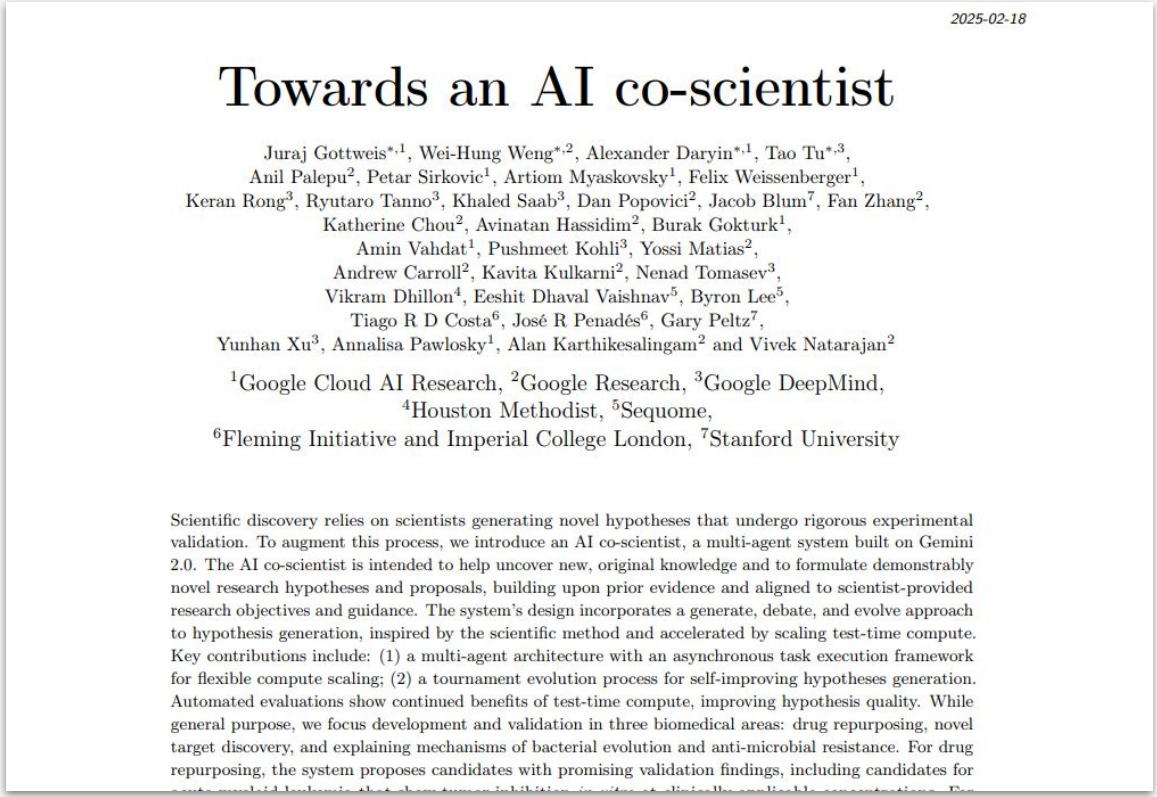
# Types of DeepAgents

# Magentic-One




https://youtu.be/RUDZZLtBo8w

# Towards an AI co-scientist



2025-02-18

## Towards an AI co-scientist

Juraj Gottweis[*,1], Wei-Hung Weng[*,2], Alexander Daryin[*,1], Tao Tu[*,3],
Anil Palepu[2], Petar Sirkovic[1], Artiom Myaskovsky[1], Felix Weissenberger[1],
Keran Rong[3], Ryutaro Tanno[3], Khaled Saab[3], Dan Popovici[2], Jacob Blum[7], Fan Zhang[2],
Katherine Chou[2], Avinatan Hassidim[2], Burak Gokturk[1],
Amin Vahdat[1], Pushmeet Kohli[3], Yossi Matias[2],
Andrew Carroll[2], Kavita Kulkarni[2], Nenad Tomasev[3],
Vikram Dhillon[4], Eeshit Dhaval Vaishnav[5], Byron Lee[5],
Tiago R D Costa[6], José R Penadés[6], Gary Peltz[7],
Yunhan Xu[3], Annalisa Pawlosky[1], Alan Karthikesalingam[2] and Vivek Natarajan[2]

[1]Google Cloud AI Research, [2]Google Research, [3]Google DeepMind,
[4]Houston Methodist, [5]Sequome,
[6]Fleming Initiative and Imperial College London, [7]Stanford University

Scientific discovery relies on scientists generating novel hypotheses that undergo rigorous experimental validation. To augment this process, we introduce an AI co-scientist, a multi-agent system built on Gemini 2.0. The AI co-scientist is intended to help uncover new, original knowledge and to formulate demonstrably novel research hypotheses and proposals, building upon prior evidence and aligned to scientist-provided research objectives and guidance. The system's design incorporates a generate, debate, and evolve approach to hypothesis generation, inspired by the scientific method and accelerated by scaling test-time compute. Key contributions include: (1) a multi-agent architecture with an asynchronous task execution framework for flexible compute scaling; (2) a tournament evolution process for self-improving hypotheses generation. Automated evaluations show continued benefits of test-time compute, improving hypothesis quality. While general purpose, we focus development and validation in three biomedical areas: drug repurposing, novel target discovery, and explaining mechanisms of bacterial evolution and anti-microbial resistance. For drug repurposing, the system proposes candidates with promising validation findings, including candidates for

https://storage.googleapis.com/coscientist_paper/ai_coscientist.pdf

# Deep Agents in the Wild (Common Applications)

**Deep Agents are designed for (mini/maxi) project-level work, not just simple queries.**

**Deep Research**

- **Task**: "Research 10 products, analyze their pricing, build a comparison spreadsheet, and write me a summary of what I should buy based on my profile."

**“Async" Coding & Development - ClaudeCode**

- **Task**: "Plan the features for a new API, implement the code across multiple files, write unit tests, and generate documentation."

# Deep Agents in the Wild (Common Applications)

**Deep Agents are designed for project-level work, not simple queries.**

**Complex Data Analysis**

- **Task**: "Load data from these financial returns files, perform a full analysis then generate a report with charts, and save the results."

**Content Creation**

- **Task**: "Research all the reviews of the DGX-Spark , create a detailed outline, write a full 3,000-word report for analysts, and save it to a file."

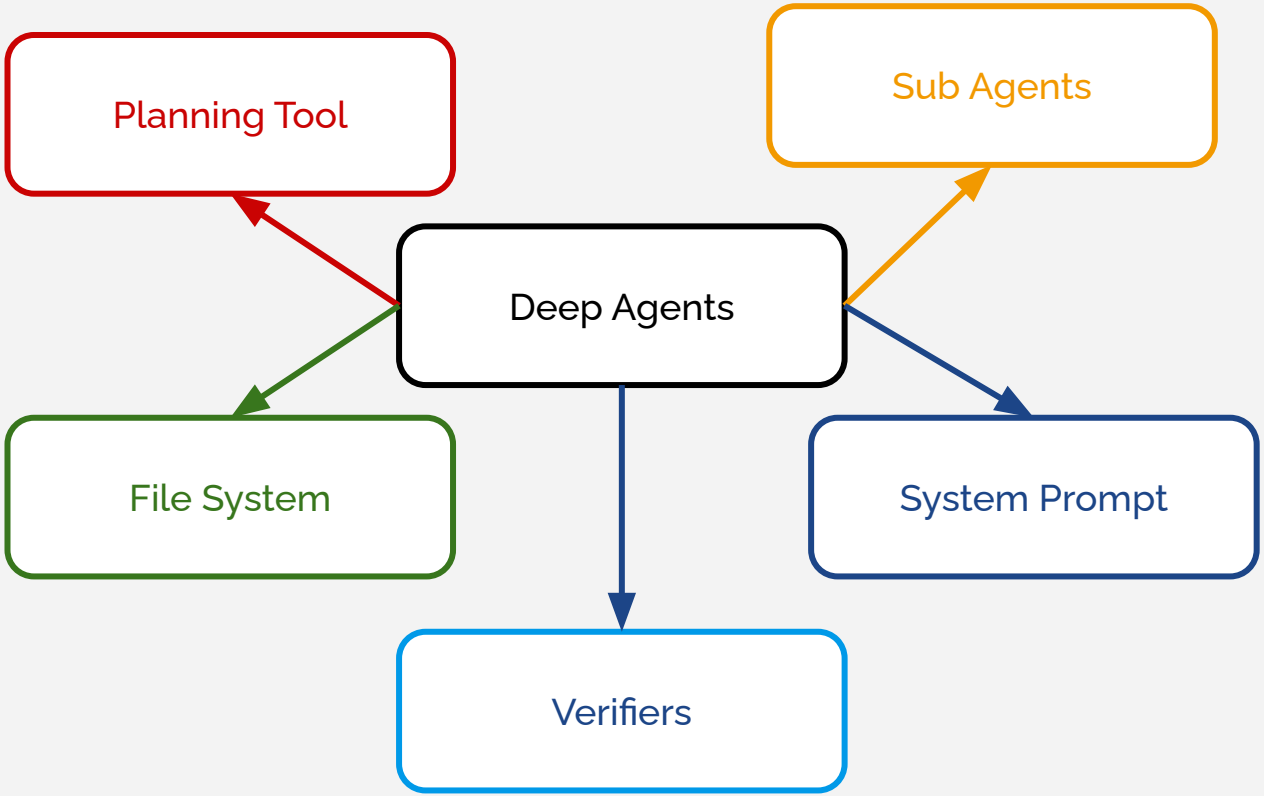
# Components of Deep Agents

# The Five components of Deep Agent Architecture

A Deep Agent's capabilities are built upon five key architectural pillars that work together to manage complex, multi-step tasks.

1. **Explicit Planning**
2. **Sub-Agents**
3. **Persistent Memory (File System)**
4. **Better Context Engineering (+ Detailed System Prompts)**
5. **Verification Systems (new)**

# DeepAgents




https://blog.langchain.com/deep-agents/

# 1. Explicit Planning

**An Actionable Plan - Often with sub plans and tracking**

Simple agents plan implicitly ("I should do X, then Y"). Deep Agents make planning an explicit, managed step.

**Mechanism:** Uses a planning tool, like write_to_dos, to create and maintain a to-do list within the agent's state.

**Function**: Between steps, the agent reviews and updates this plan, marking tasks as pending, in_progress, or completed. It's a context engineering strategy to keep the agent on track.

# 1. Explicit Planning

**Impact**: If a step fails, the agent doesn't just retry. It updates the plan to accommodate the failure, enabling robust recovery and keeping the agent focused on the high-level goal.

# Planning Prompts Examples

# 2. Sub-Agents

**The Orchestrator → Sub-Agent Pattern**

Complex tasks require specialization. Instead of one "jack-of-all-trades" agent, Deep Agents delegate work.

**Mechanism**: A main "Orchestrator" agent spawns specialized sub-agents (e.g., "Researcher," "Coder," "Data Analyst") for specific tasks.

**"Context Quarantine"**: Each sub-agent gets a fresh, clean context with only the specific task it needs to perform. The main agent's history is wiped from its view.

# 2. Sub-Agents

**Impact**: This prevents the main agent's context from being polluted with low-level details. The sub-agent performs its work (including retries and errors) and returns only the final, synthesized answer.

# 3. A File System (Persistent Memory)

**Shifting From "Remembering Everything" to "Knowing Where to Find It"**

To overcome context window limits, Deep Agents offload their memory to an external source or an in memory datastore.

**Mechanism**: Agents are given tools (read_file, write_file, edit_file) to interact with a virtual file system (often a dictionary in the agent's state).

**Function**: Intermediate results, raw data, code snippets, and research notes are written to files. Subsequent steps or sub-agents reference these files instead of re-populating the context window.

# 3. A File System (Persistent Memory)

**Impact**: This keeps the context window clean and focused on the immediate task, enabling the agent to handle vast amounts of information over long periods. It's also more scalable and avoids concurrency issues.

# 4. Better Context Engineering

**Smarter Models Don't Require Less Prompting—They Require Better Prompting**

Deep Agent behavior doesn't emerge from a simple "You are a helpful AI" prompt. It's meticulously engineered.

**Mechanism**: Uses highly detailed system prompts, sometimes thousands of tokens long, that act as an operating manual for the agent.

# 4. Better Context Engineering

**What it Defines:**

- **Protocols for when to plan vs. act.**
- **Rules for spawning a sub-agent.**
- **Detailed tool definitions with examples.**
- **Standards for file naming and directory structures (or data stores).**
- **Strict formats for output and collaboration.**

**Impact**: This detailed instruction set is the foundation that guides the agent to effectively use its other three pillars.

# 5. Verification Systems

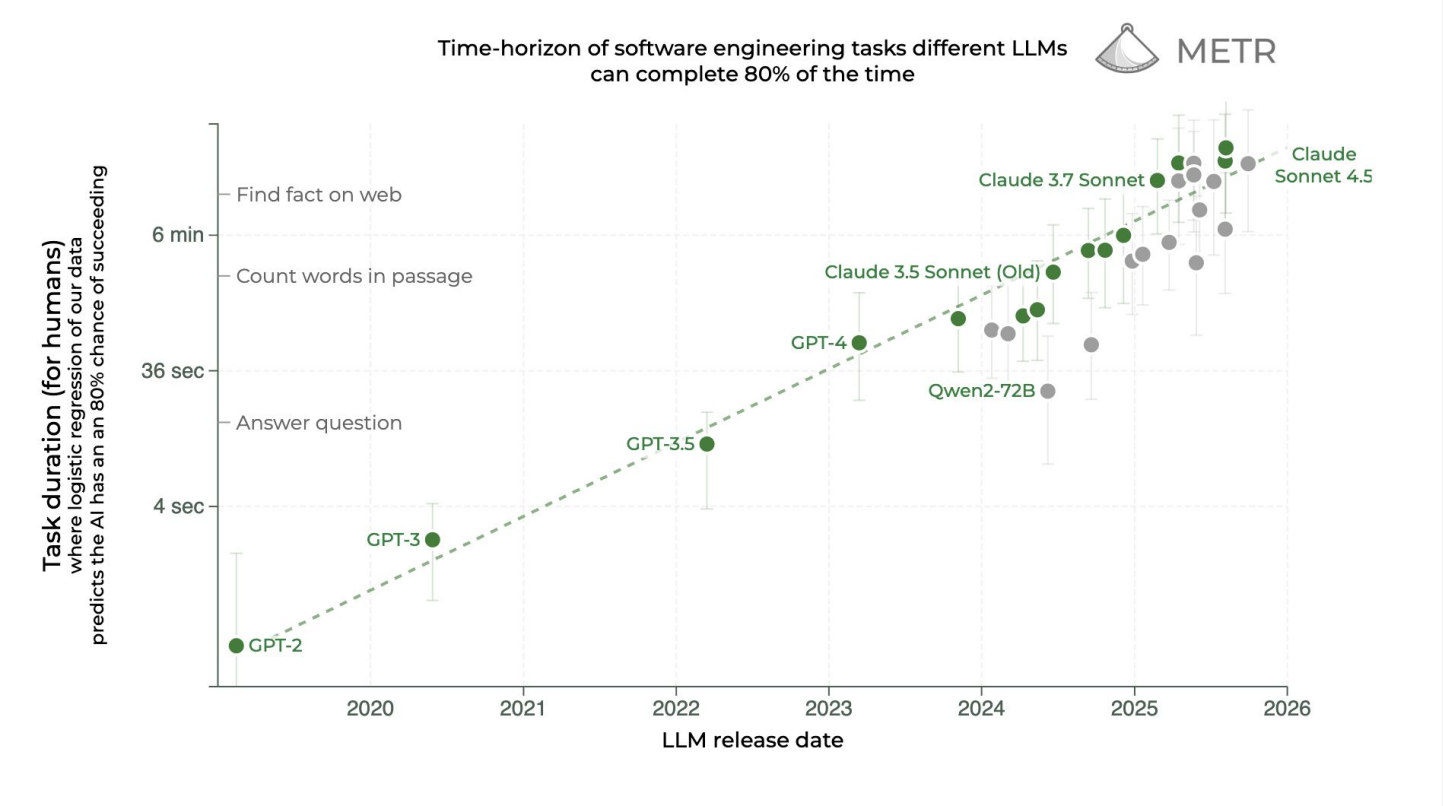
**Verifying outputs allows for better trajectories**

Deep Agent verification allows the model and the agent to stay on track with the overall goal and plan .

**Mechanism**: Agent verification can be done in many different ways, from things like solvers, through to code compilation, through to checking systems.

# Long Running DeepAgents

# Sustaining Long-Running Tasks



# How Deep Agents Sustain Long-Running Tasks

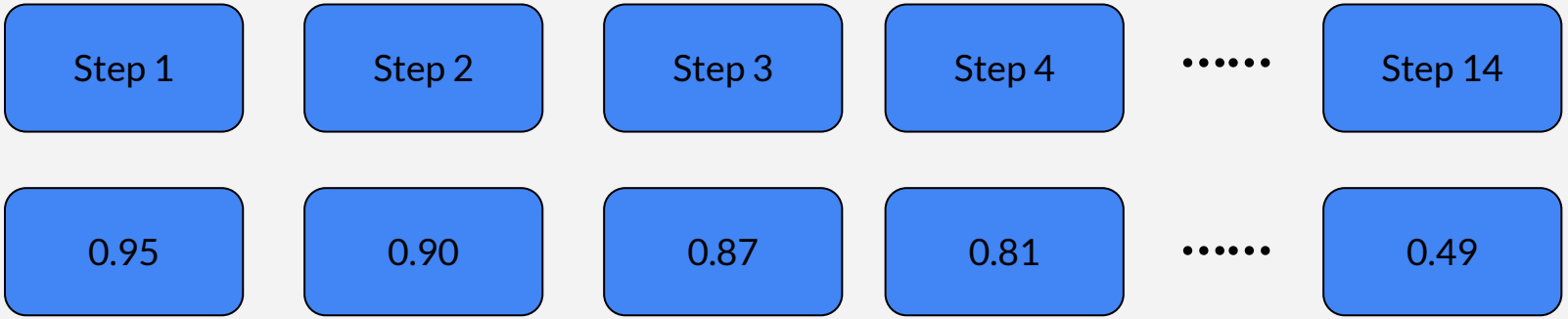
The five components work together to overcome the limits of shallow agents.

- **Planning Tool Prevents Goal Loss**
  - The to-do list acts as a constant, high-level reminder of the overall objective, keeping the agent on track.
- **Persistent Memory Prevents Context Rot**
  - By writing intermediate results to a virtual file system, the agent keeps its context window clean and focused on the current step.

# How Deep Agents Sustain Long-Running Tasks

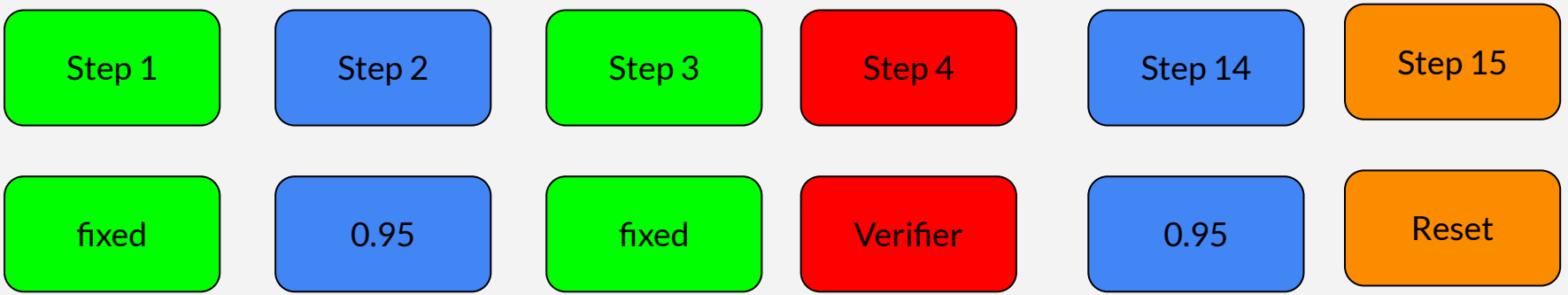
- **Sub-Agents Manage Complexity**
  - Delegating complex sub-tasks isolates them, preventing the main agent's context from being polluted by details.
- **The Plan Enables Recovery**
  - If a step fails, the agent doesn't just retry blindly. It can update the plan to accommodate the failure and try a new approach.

# Agent Trajectories




- The more autonomous steps your agent takes the less the chance it will work

# Desired Agent Trajectories

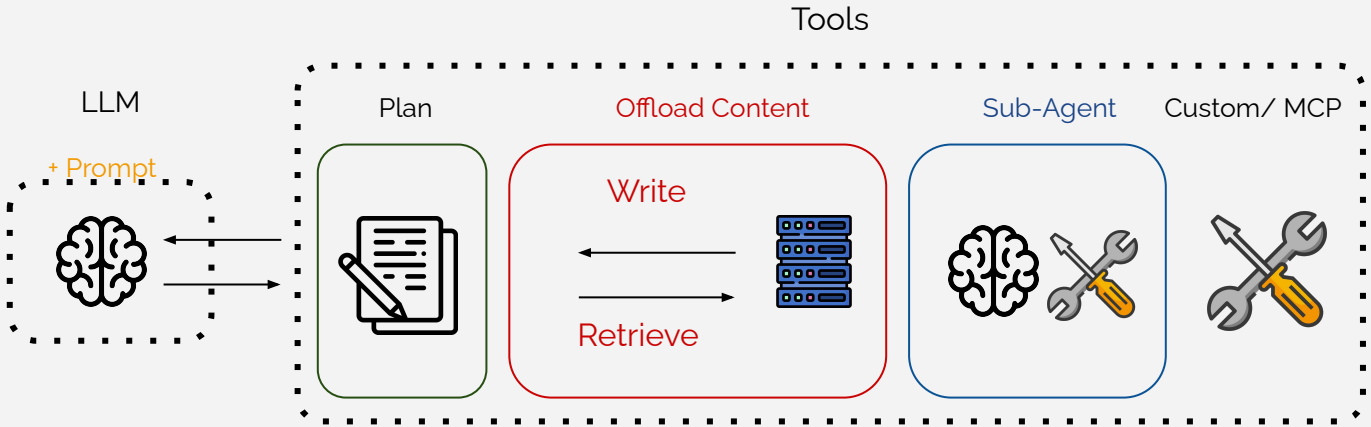


# The Key Context Engineering Concepts

- **Context Rot**
  - Gradual drift where the agent's working context accumulates stale, contradicted, or paraphrase-distorted facts over long trajectories
  - More tokens degraded performance
- **Context Compaction**
  - Token-budget pruning of candidate context
  - Compressing the context (replacing tool calls etc)
  - Context Summarization
- **Context Isolation**
  - Context retrieval (often via RAG)
- **Context Off Loading**
  - Saving Context off to a file system

# LangGraph DeepAgents

# Deep Agents Abstraction




https://blog.langchain.com/deep-agents/

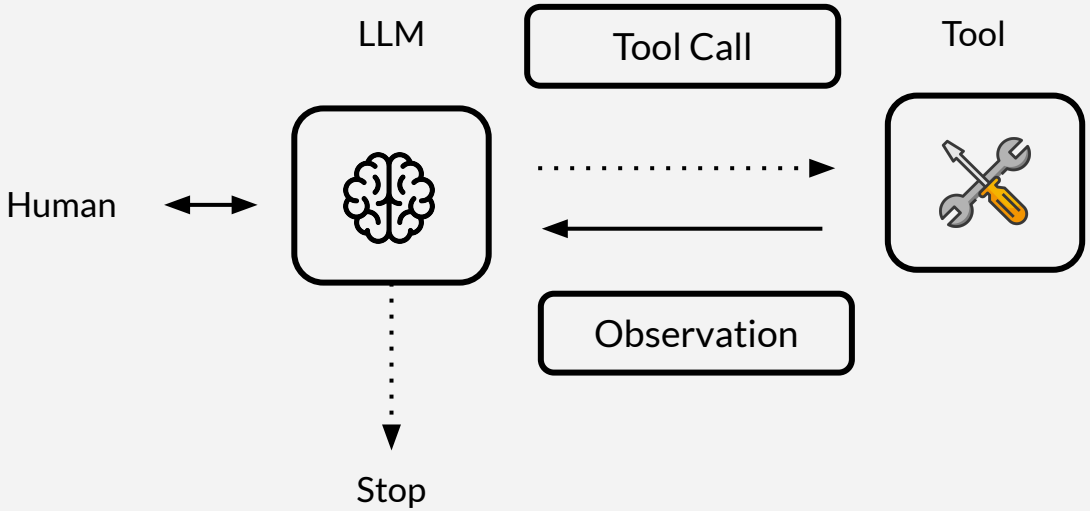
# LangGraph is On Rails

- The more you limit the choices the less things can go wrong

# The Technical Foundation: DeepAgents + LangGraph

- The deepagents Python library makes this architecture accessible.
- An open-source package to easily build and customize your own Deep Agent.
- Fundamentally built on top of LangGraph, which serves as the agent runtime.
- In LangGraph, agents are represented as graphs, allowing for complex control flow, loops, and state management.
- The core algorithm uses LangGraph's create_react_agent wrapper.

# ReAct Agent




**ReAct: Synergizing Reasoning and Acting in Language Models - https://arxiv.org/abs/2210.03629**

# Managing State: The DeepAgentState Object

- State is managed explicitly, not just in conversation history.
- DeepAgentState is a custom object that tracks all information throughout the task.
- Key Attributes:
  - **Messages & State**: The standard list of conversation history.
  - **to-dos**: The agent's plan, a list of tasks with statuses ('pending', 'in_progress', 'completed'). Managed by the planning tool.
  - **files**: A dictionary representing a virtual file system (filename: content). This is the agent's persistent memory.

# Code Examples

# Questions?

sam@reddragon.ai
@sam_witteveen