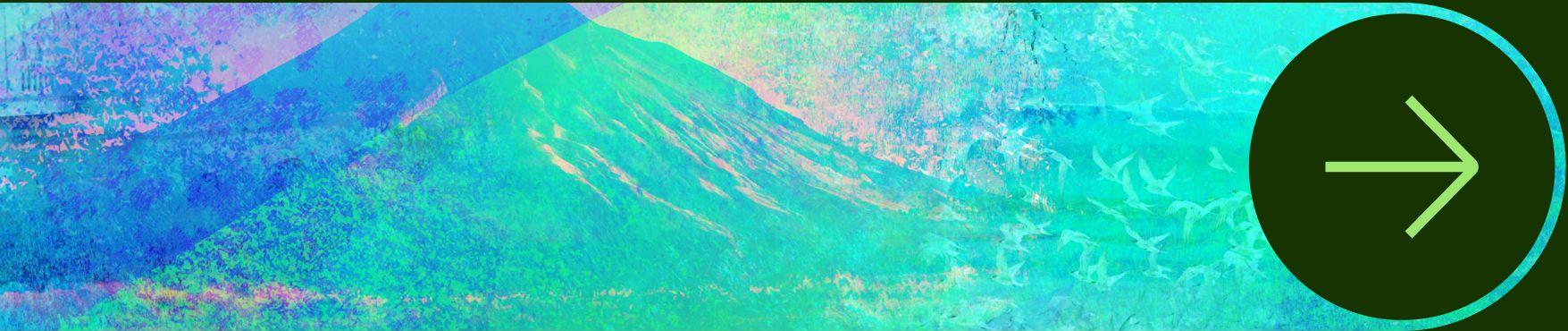


COLLABORATING AGENTS: MULTI-AGENT SYSTEMS AND WHEN TO USE THEM



Dr Egor Kraev, Head of AI, Wise
egor.kraev@wise.com



What is this workshop about

- The hype around multi-agent systems keeps growing
- Hardly a week passes without another agent framework being announced
 - I am also guilty of one - motleycrew.ai
- Each seems to do things slightly differently
- How do you navigate this complexity?

It's all very simple really

- There is only a handful of basic patterns for agent interaction
- Many of the new patterns are just old patterns, but with an LLM inside
- This workshop will walk you through the different patterns and reasons you might want to use them - or not



WAYS TO USE AN LLM

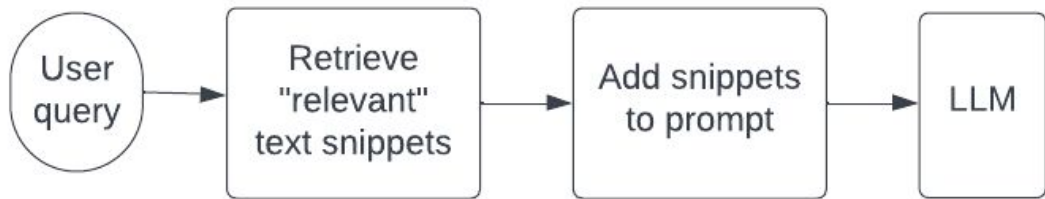


Straight call into an LLM

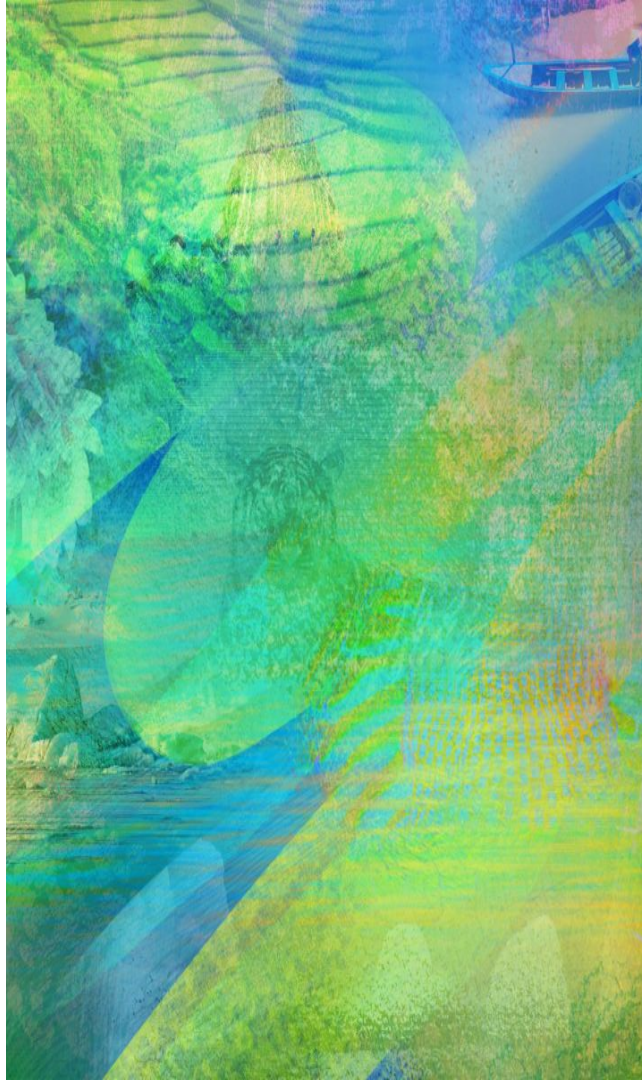
- Text (and possibly images, sound) in → text (and possibly images etc) out
- **Problem 1:** An LLM only aware of the data used during its training, so **no current events or internal documents**
- **Problem 2:** Even though prompt size is no longer a formal limitation in practice, LLMs so far **struggle to extract information** well from **large prompts**

Pre-retrieve relevant data from web or internal storage, **add it to the prompt**, feed it to the model in a single call

This allows the LLM to answer questions about **data never seen in training**



Retrieval-Augmented Generation (RAG)



RAG challenges

- Have to **carefully select** the documents to avoid over-large prompts
- Many, many possible tricks for selecting the documents, no good standard solution yet
- What if the answer is that you should now **look for other information elsewhere?**



Enter LLM agents



Yes that picture was created using ChatGPT

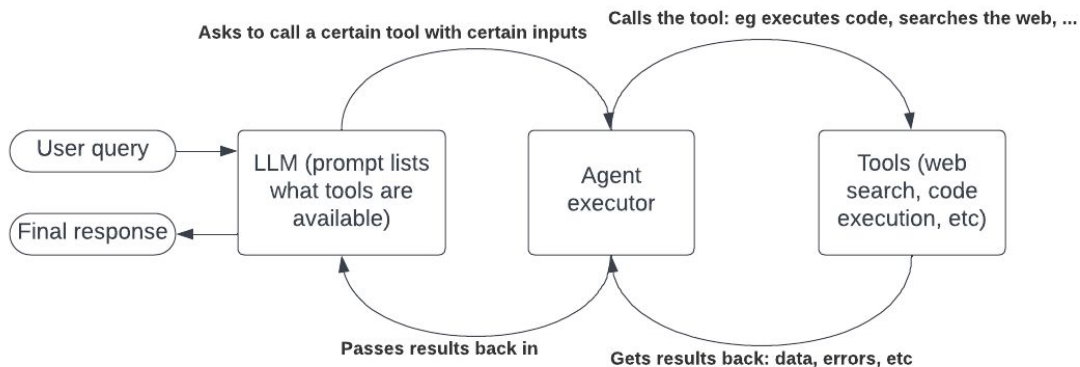


Agents

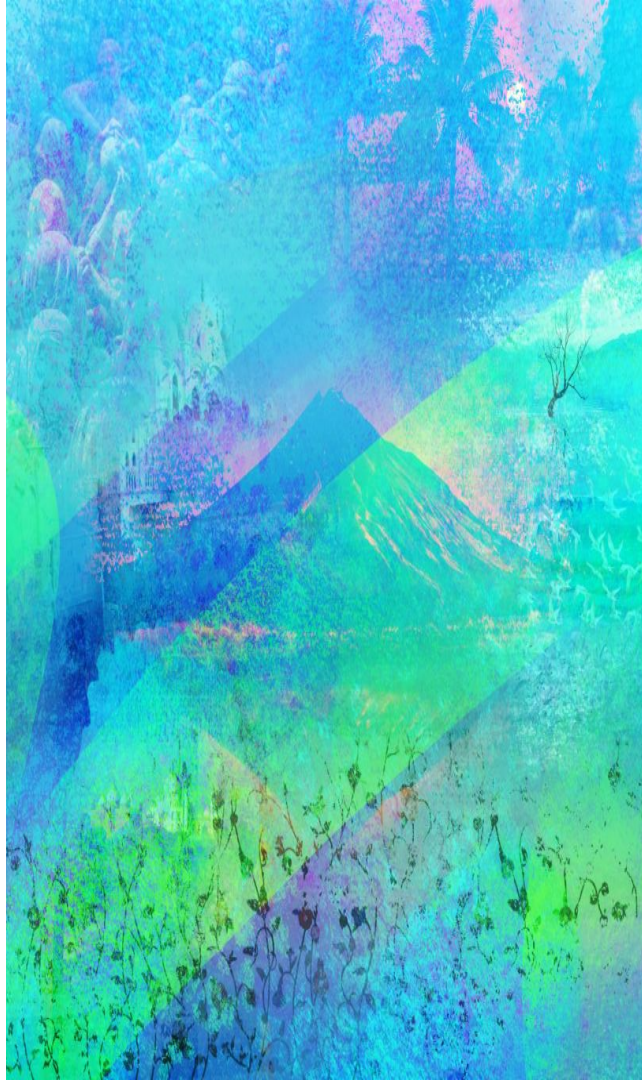
An LLM agent is some sort of a **loop** that can use an LLM's outputs to **call other software and fetch additional data**, and **feed the results back** to the LLM, repeating that cycle until an objective is achieved

The definition is vague because it can be (and is) implemented in quite different ways by different frameworks

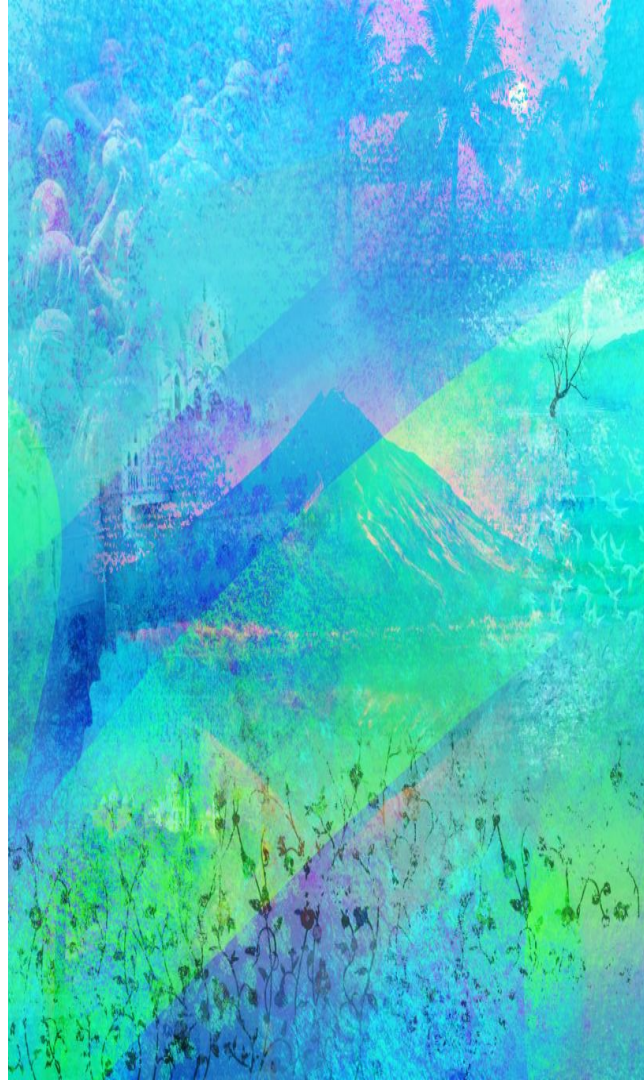
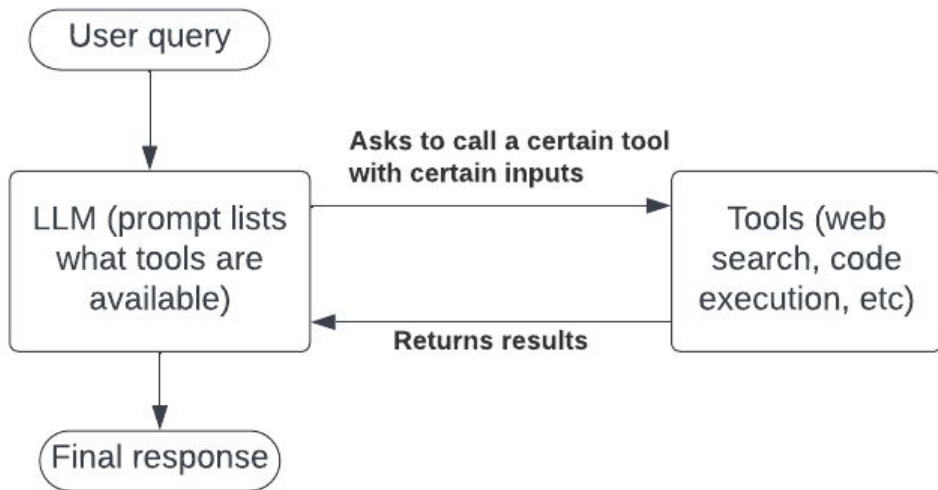
Basic agent pattern



- Tools: ways to interact with rest of the world
- Agent: an LLM whose prompt tells it what tools are available. Can ask to use them (eg to run code it has generated)
- Agent executor: Has access to the tools that LLM is aware of, calls them when the LLM asks, and passes results back

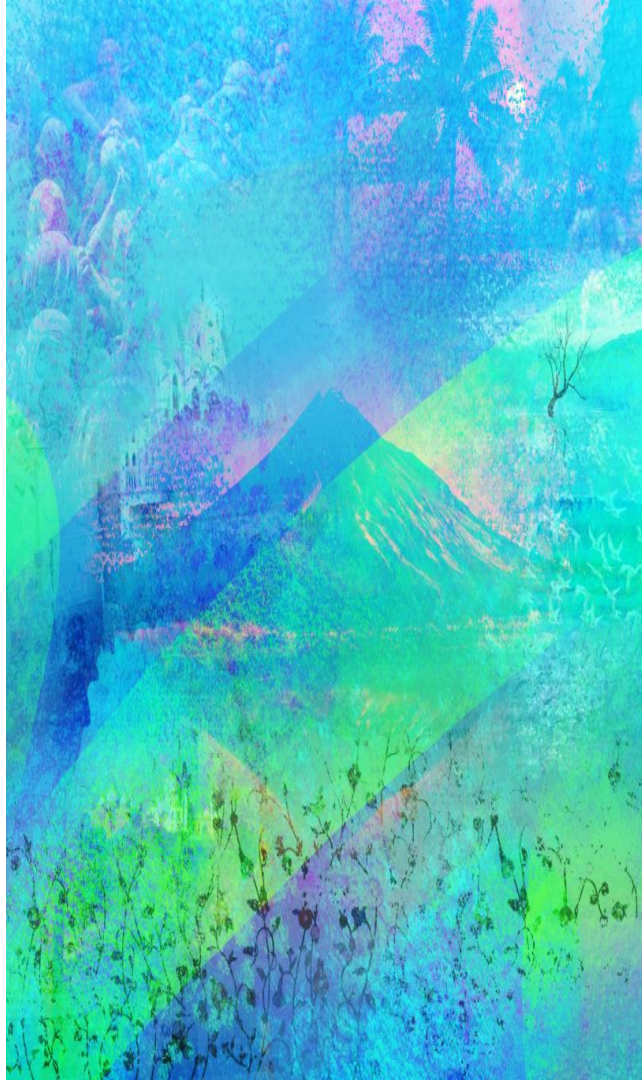


Simplified representation of tool-calling pattern




Major agent types

- **Planning**
 - First list the steps to take, then do them one after another
- **ReAct**
 - A “reflect - act - observe results” loop
- **Chain of Thoughts**
 - Reason about the problem step by step
- **Tree of Thoughts, Graph of thoughts, ...**
 - Build/walk graphs of reasoning steps, until solution reached



**WHEN WOULD
YOU USE AN
AGENT?**

An abstract, textured background image featuring warm colors like orange, yellow, and red. It includes faint, overlapping architectural elements such as a classical column and a dome, suggesting a historical or classical setting. The overall style is painterly and layered.

When you're not sure ahead of time what tools you'll need to call, with what inputs

Agents are useful when you need to first evaluate the incoming (potentially RAG-enriched) inputs and then **do further steps based on the outcome**

- Multi-purpose agents that can use multiple tools, depending on what the user asks for
- Web searches for the terms that were only determined from the enriched context
- LLM converts user question into SQL, tool runs the query
- Math operations on intermediate outputs



Ask for more information dynamically

- LLM gets a question within the prompt, together with some context, then calls a retrieval tool with additional queries derived from these
- As opposed to hard-wired enrichment in the basic RAG pattern
- Example: multi-step research agent

ADVANCED TOOL USAGE PATTERNS



Agents can be used as tools

Agent using another agent **as a tool**

- Parent agent only gets the final result of the subtask it gives to the tool agent
 - Smart tool/LLM tool
 - Writer-critic
 - Big picture vs detail
 - Validation loop

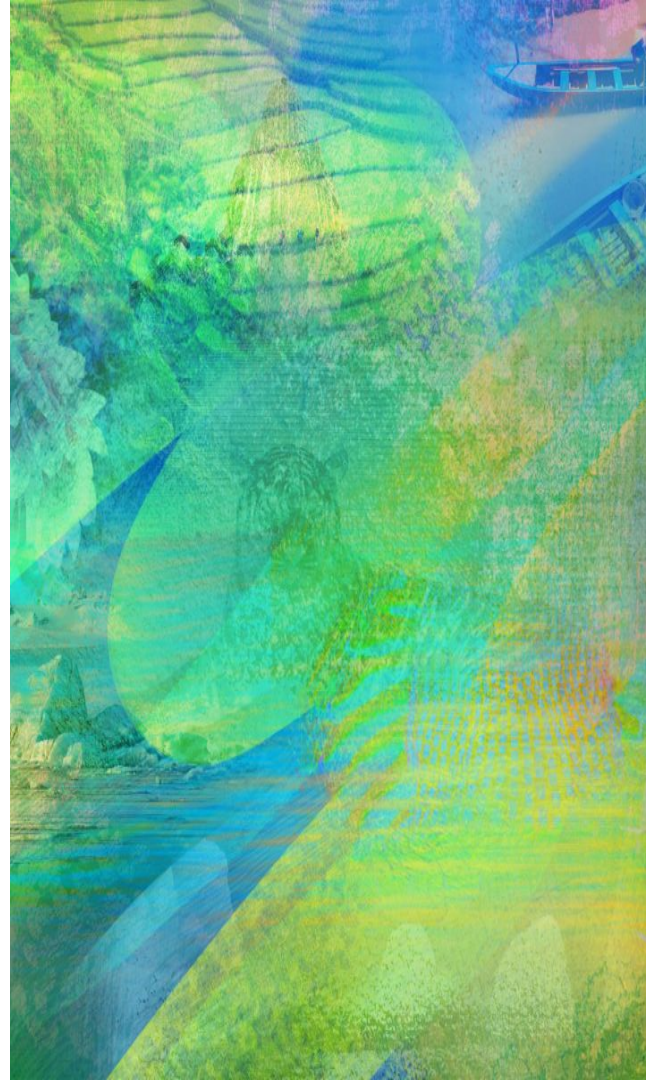
Smart tool

- **Specializes** in a task that itself needs a dedicated prompt, for example
 - Image generation
 - Text retrieval based on reasoning rather than similarity



Writer - critic

- One agent **writes text or code**, another one **critiques it** and gives improvement recommendations, **iterating** if needed
- Can have **multiple critics** if want to optimize the text according to several criteria





Big picture vs detail

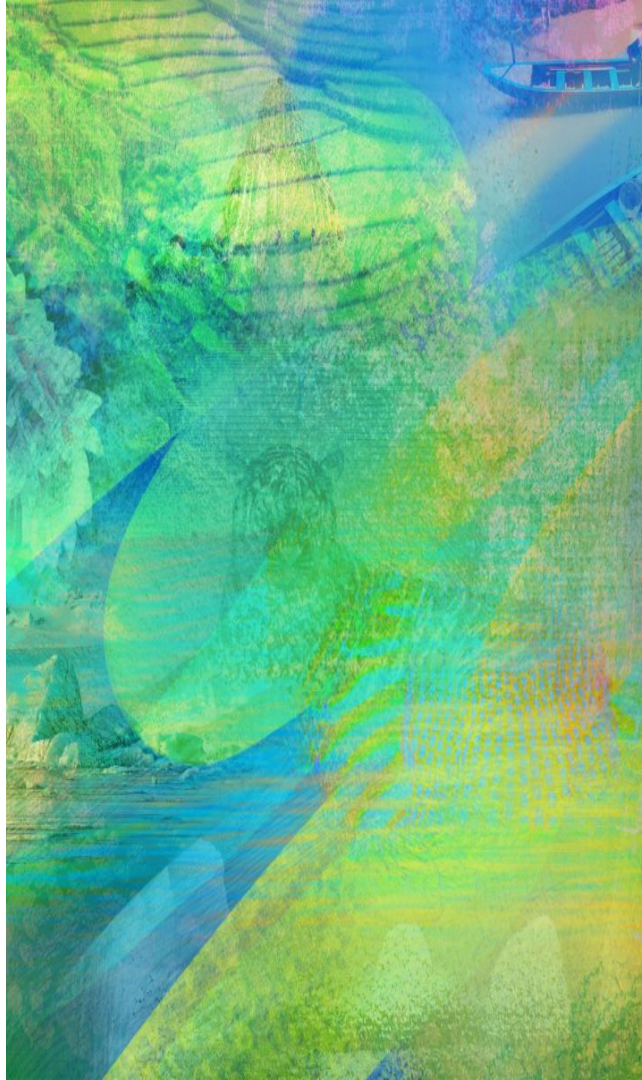
An agent **tracks the master plan, delegates subtasks** to execution agents, for example

- Story outline vs plotting of individual chapters writing of paragraphs in a consistent style
- Overall software project plan and diagrams vs writing individual bits of code

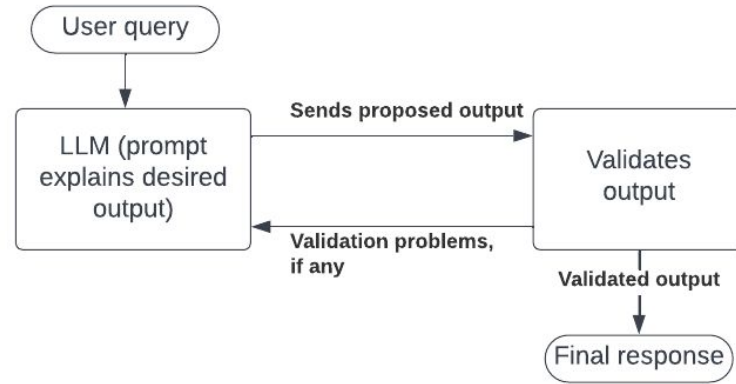
Validation loop with feedback

Another frequent use for agents is when the output might need **several iterations** to get right.

- LLM writes code to solve a problem
- Has it executed by a PythonREPL tool
- Gets back the results, including errors
- In case of errors or bad results, tries again
- Once code runs as planned, returns

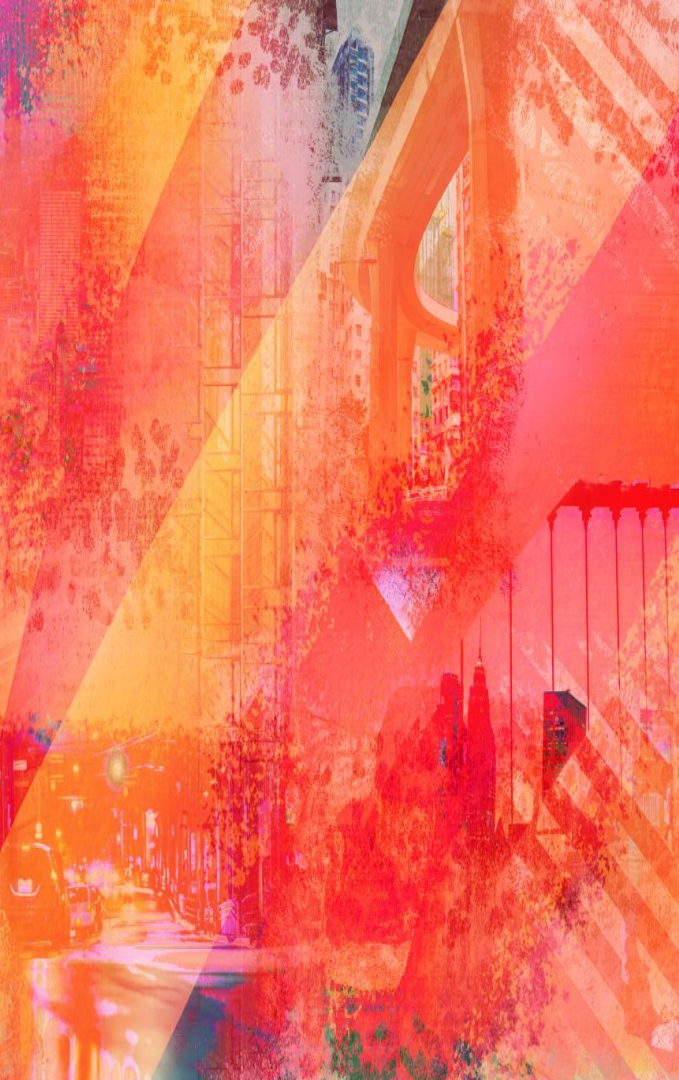


Forced validation pattern



- From the point of view of the agent, looks like tool calling
- Tool validates inputs and returns them **directly** if successful
- If validation fails, reasons are returned to the agent
- Agent can **only** return results via the tool call
- Natively implemented in motleycrew.ai

**WHEN IS AN
AGENT NOT
ENOUGH?**



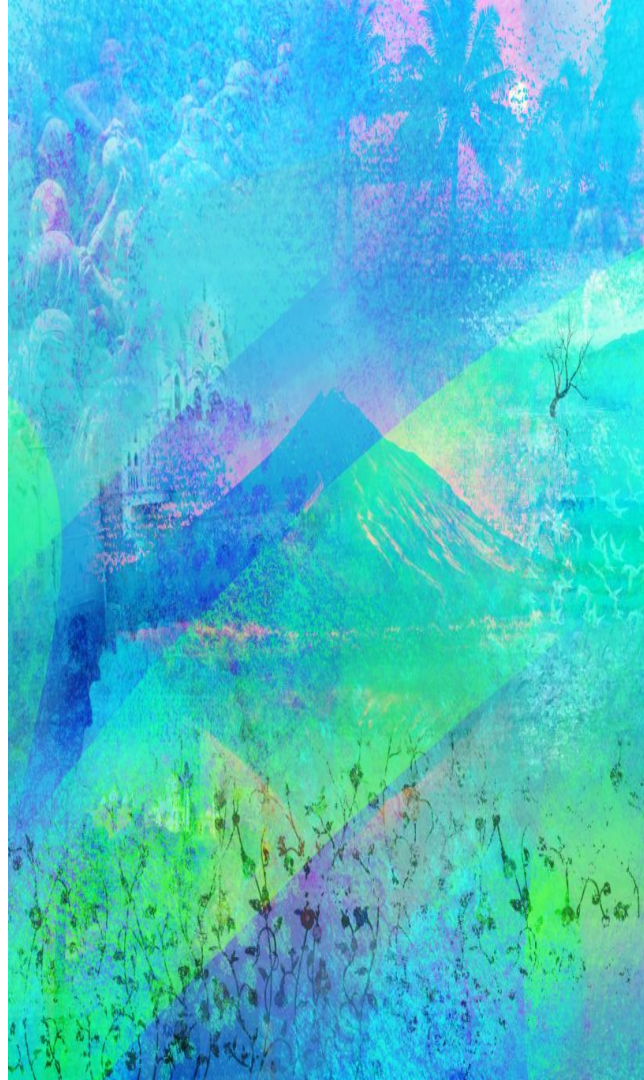
Prompt engineering vs flow engineering

LLMs have trouble dealing with **too many different instructions** squeezed into **one prompt**, the trick to making them work is splitting the task into many small, logically simpler steps, and run them in the right order

This is known as flow engineering (see [AlphaCodium on GitHub](#) for details)

Example: contract evaluation

- We at Wise needed to build a tool to evaluate the **compliance of vendor contracts** with our **guidelines**
- Some two dozen guidelines to evaluate each contract against
- With **all guidelines and the whole contract in a single prompt**, it **ignored half the guidelines**
- When we did a for-loop over the guidelines, GPT-4 'found' violations of each one (**eagerness**)
- When evaluating a single guideline against the whole contract text, it made **silly mistakes** such as saying $3% > 7%$



Example: how we solved it

- First do a **for loop** over **all the guidelines** asking whether a given guideline is even **relevant**
- Then a loop over all **relevant** guidelines, **just extracting** the text chunks LLM thinks are in violation
- Then for each text chunk feed **just the chunk and the guideline** into the prompt, **re-assess violation**
 - This **performed much better** than the same task when feeding in the whole contract text
- Then for the chunks deemed in violation, **generate proposed wording**

