Design Patterns for Compound AI Systems (Conversational AI, CoPilots & RAG)

15 min read · Mar 17, 2024

Raunak Jain Follow

Listen Share More

How to build configurable flows and compound AI systems using open source tools.

What are compound AI systems?

Recently, researchers at Berkeley wrote an article — **The Shift from Models to Compound AI Systems,** where they compiled the progress on LLM apps and stressed on the evolving nature of complex pipelines of components working together to build intelligent systems, rather than a closed singular model. Although, the resulting system might use the same underlying model (like GPT4) but the prompting and context warrants it to be called different components.

Some common deployment patterns of these systems:

1. **RAG (*retrieval* and understanding is key)** — with access to Thought generation, Reasoning and contextual data, these systems self-reflect and try to understand a user's query in advanced ways, before responding back with an answer, ideal set up is an **Agent Assist** system. When combined with user facing models / systems like a Dialogue model, a RAG system could be a part of a Conversational AI or a CoPilot system.

2. **Multi — Agent Problem Solvers (*collaborative* role playing is key)** — these systems rely on collaborative and automated build up of solutions based on outputs of agents being fed into each other with a well defined role and purpose. Each agent has access to it's own set of tools and can assume a very specific role while reasoning and planning it's actions.

3. **Conversational AI (*dialogue* is key)** automation softwares like **customer service agents**, which interact with humans within an app / ecosystem and execute repeatable tasks based on inputs and validations from humans. The most important aspect here is conversational memory and dialogue generation with the feeling of having a conversation with a human. The Dialogue Model can have access to an underlying RAG system, or a **Multi-Agent** problem solver.

4. **CoPilots (a *human in the loop* interface is key)** — with access to tools, data, reasoning and planning capabilities, and specialized profiles, these systems can independently interact with a human while solving problems with closed solutions. The key differentiator for becoming a CoPilot is understanding of the environment in which the human works. For e.g. MetaGPT Researcher: Search Web and Write Reports, A measured take on Devin, Let's build something with CrewAI, Autogen Studio.

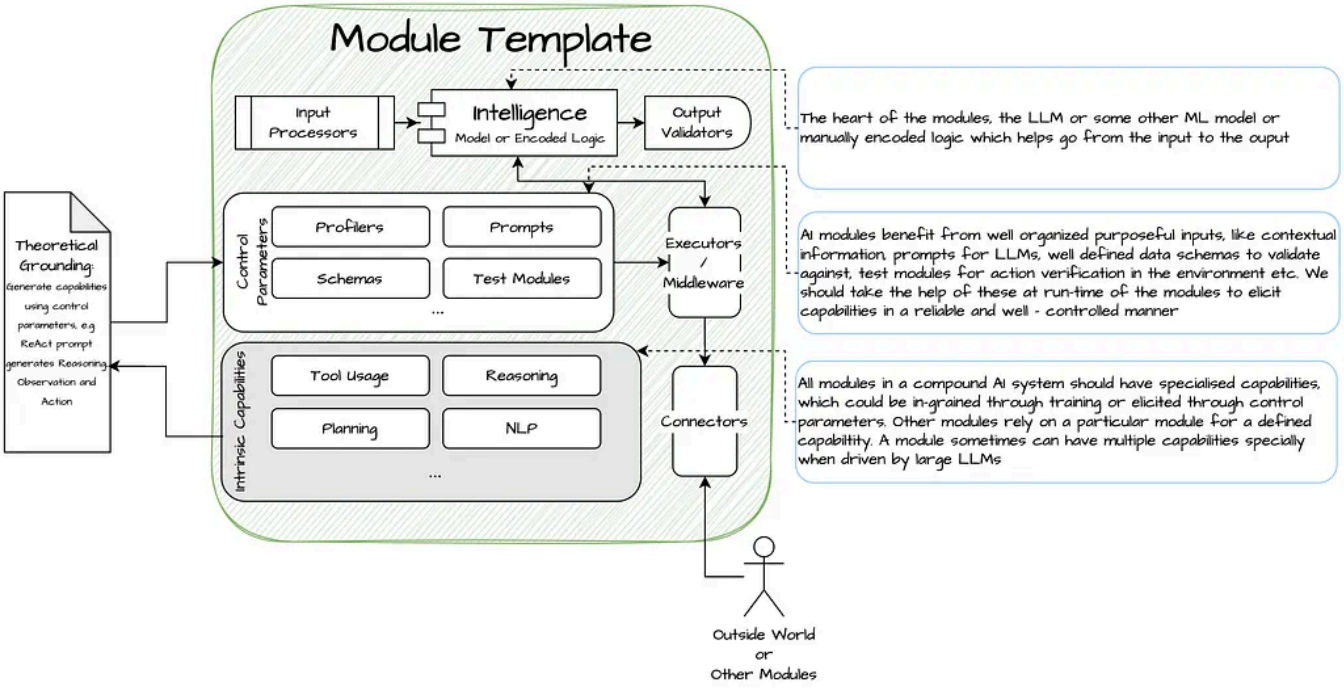
> **Cautionary note:** based on the evidence I have after deploying several systems using LLMs, I can safely say these systems are not the silver bullet everyone wants them to be. Just like any other AI system, you need a lot of engineering around an LLM to make them do basic things, and even then they can not be reliable and scalable with performance guarantees.
>
> **Promising path:** The biggest value add in using LLMs in your ecosystem is to enable Machine Learning, i.e. understand where your product or experience is weak and feed the data back into the system in a manner which improves performance in the long run. Imagine replacing or reducing the work of human annotators. But in closed systems where we can test and evaluate LLMs output with, it can surprisingly do a lot! but at a big cost..but also, that generates data for building better LLMs :) This feedback effect which Machine Learning thrives on, is the biggest value add from LLMs.

Components of these systems and how they interact with each other to build complex systems

Compound AI systems are usually deployed using interconnected "modules" which rely on each other to do certain tasks and chain together to execute design patterns.

Here, **modules** are referred to those individual components in the system which do a well defined task with or without the help of an underlying system like Search Engine, LLM etc. Some common modules include generator, retriever, ranker, classifier etc, traditionally called tasks in NLP. These are conceptual abstractions

which are domain specific (e.g. NLP modules might have different abstracted modules than Computer Vision or Recommendation systems, but they might all rely on the same underlying model serving or search provider).



Key components of a module

Some other generic modules

In pop AI culture, we have become used to terms like Tools, Agents, Components, these can all be thought of as modules.

LLM based Autonomous Agents — a key module in compound AI systems

One form of a module is an **Autonomous Agent,** which can reason and plan in an autonomous manner with the help of an LLM. An autonomous agent might depend on a list of sub-modules to decide how to reason and plan actions while interacting with the environment.

source

Key capabilities or skills of a Autonomous Agent module:

Reasoning capabilities, leading to a chain of thought, resulting in a plan of action.

1. **Reasoning** — logical methods to solve a problem by making observations, generating hypothesis and validating based on data etc.

2. **Thought** — an application of reasoning and generation of a coherent cause-effect relationship.
3. **Chain of thought** — a sequence of thoughts which break a solution into a sequence of logically connected reasonings.
4. **Planning —** decision making skills to do sub-goal formation and build a path from a current state to a future state. This usually benefits from having access to the underlying environment to take action and learn from. Read more about LLM planning here.
5. **Tools** — sub modules which allows an agent to interact with the external world based on instructions from the agent.
6. **Action** — as the name suggests, it is a decisive step which an agent takes in the pursuit of a goal, through a plan. A tool might be invoked to trigger an action. Actions are made in the environment.
7. **Environment** — the external world in which the actions and rewards for the agent originate, like a custom app (like Microsoft Word, Coding Environment) or a game, or a simulation etc.

Are we working with Stochastic Parrots?

> Note: although there is a lot of theoretical work around the lack of reasoning and planning capabilities of LLMs, see this and this, but it is pretty evident that LLMs can parrot a "solution methodology" capably. That means, if given a problem to solve, and some examples of how it was solved previously, an LLM can replicate the logical thought process well, if it generalises or not is not what we are going to concern ourselves with here, but as Yann Le Cunn says, Autoregressive models are doomed!
>
> While in the enterprise setting, all we need is to repeat tasks reliably and learn from previous experiences without being too creative.

How do these parrots actually plan?

Based on this survey, LLMs seem to be able to empower Autonomous Agents by doing the following well:

1. Task decomposition — Tasks in real life are usually complicated and multi-step, bringing severe hardness for planning. This kind of method adopts the idea of

divide and conquer, decomposing the complicated into several sub-tasks and then sequentially planning for each sub-task e.g. TDAG

2. Multi-plan selection — This kind of method focuses on leading the LLM to "think" more, generating various alternative plans for a task. Then a task-related search algorithm is employed to select one plan to execute. e.g. ReWoo
3. External module-aided planning — also plan retrieval.
4. Reflection and Refinement- This methodology emphasizes improving planning ability through reflection and refinement. It encourages LLM to reflect on failures and then refine the plan. e.g. Self critique and refinement loops.
5. Memory-Augmented Planning — This kind of approach enhances planning with an extra memory module, in which valuable information is stored, such as commonsense knowledge, past experiences, domain-specific knowledge, et al. The information is retrieved when planning, serving as auxiliary signals.

Understanding the planning of LLM agents: A survey

How do we train for these capabilities, specially for RAG? See Fine-tuning LLMs for Enterprise RAG — A design perspective, for a deep dive.

With these module abstraction in our head, let us see how different design patterns can be developed to solve complex problems in Conversational AI, RAG and CoPilot systems.

Design patterns of compound AI systems

Some clarifying definitions

Due to the "pop culture" like environment around AI right now, there are mis-used and mis-understood terminologies, which forces me to clarify a few things before proceeding:

1. What is an "**Agentic**" pattern?
   The core benefit of an **Autonomous Agent** is that it decides what steps to take on it's own. If we have to manually define a flow or decisions, it is just an intelligent workflow. But if the flow is un-defined and the decision making is enabled by the use of capabilities mentioned above along with tools and actions, it is an "Agentic" pattern. For e.g., Agentic RAG is a pattern where a module is given access to search tools and it produces complex search flows automatically without any pre-defined steps.

2. What is a "**workflow**"?
   In simple terms, a workflow is pre-encoded and manually declared plan which solves problems in a predictable and repeatable manner.

3. What is "**multi-agent**"?
   A system where different modules, assuming different roles and responsibilities, interact with each other to work upon each other's outputs and solve a problem collaboratively.

When building an agent, some of the key things to optimize for are:

1. Agent **Profiling** — how the agent should behave, mostly prompt driven and highly dependent on "role" profiling.

2. Agent **Communication** — common patterns of communication between these modules can be abstracted as in the image below.

3. Agent Environment **Interface** — important to collect feedback from the execution environment to learn, adapt and ground the generation.

4. Agent **learning** and evolution — agent systems learn from interactions with the environment (specially for coding assistants), other agents, self-reflection, human feedback, tool feedback etc. We will explore what strategies each architecture uses while solving problems in real time.

Considerations before choosing a pattern

To abstract away and define design patterns which allow for the development of RAG, Conversational AI, CoPilots and Complex Problem Solvers, we need to ask the following questions:

1. Is the flow between modules well defined vs autonomous? *Engineered flow vs Agent System.*

2. Is the flow directional or “message passing” inspired? Is it co-operative or competitive? *Agent Modulo*.

3. Is the flow self learnable? Is self-reflection and correction important?
   - Can we have reasoning and action taking loops?
   - Can modules feed from each other?

4. Is each module’s output testable in the environment?

5. Is the execution of the flow stateful and/or change with user’s input?

Deployment pattern 1 — RAG / Conversational RAG

The diagram below shows main responsibilities of modules within a RAG / Conversational RAG system. This has been traditionally an IR domain which was first improved by Neural Search, Knowledge Graphs and then generative loop based approaches using LLMs. Conversational IR is another view of this system when IR and Dialogue systems merge and queries are viewed as contextual objects which transform during a conversation.

For the success of a RAG system, it is key to understand the query of the user and map it to the underlying knowledge (structured or unstructured) and feed it to the Generator / Dialogue Manager with appropriate instructions. All of this can be executed using a well-defined workflow, or using Agent modules, which decide what steps to execute dynamically (to be elaborated more in the next section).

A RAG flow, with handoff to Dialogue Manager — If Dialogue Manager is an Agent, RAG could be a tool

Let’s look at some intermediate modules / tools, which allow the agent to navigate this complex world of RAG.

Query Understanding and Reformulation

Query expansion / Multi — query

Using LLMs to expand query improves search results when sparse and statistical retrievers are used.

Query rewriting / Self — Query

A self-querying retriever is one that, as the name suggests, has the ability to query itself. Specifically, given any natural language query, the retriever uses a query-constructing LLM chain to write a structured query and then applies that structured query to its underlying VectorStore. This allows the retriever to not only use the user-input query for semantic similarity comparison with the contents of stored documents but to also extract filters from the user query on the metadata of stored documents and to execute those filters.

Entity recognition

Query Enrichment

Knowledge or Intent retrieval

Multi — Document Search

Dialogue Management

Response Generation

Agentic RAG

Agentic RAG is a design pattern where a module, powered by an LLM, reasons and plans how to answer a question based on the set of tools available to it. In advanced scenarios, we might also connect multiple agents to solve RAG in creative ways where agents not only retrieve, but verify, summarize etc. See the Multi — agent section for more on this.

The key steps and components which need to be refined:

1. Plan based on reasoning, subtask formulation, and systematic arrangement.
2. Self correction based on self-consistency, due to generation of multiple paths and reasoning, planning based RAG approaches (ReWoo and Plan+) work better than just reasoning based (ReAct).
3. Ability to adapt based on execution, a more multi-agent paradigm.

Usually, these are executed using the following patterns:

Reasoning based Agentic RAG

ReAct

Reason and Act using search tools

Planning based Agentic RAG

https://blog.langchain.dev/planning-agents/

ReWoo

ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models

Augmented Language Models (ALMs) blend the reasoning capabilities of Large Language Models (LLMs) with tools that allow...

arxiv.org

ReWoo — leading to a much lower token generation than ReAct.

> See this for more details on why ReAct is much worse than ReWoo.

PlanRAG

It consists of two components: first, devising a plan to divide the entire task into smaller subtasks, and then carrying out the subtasks according to the plan.

Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models

Large language models (LLMs) have recently been shown to deliver impressive performance in various NLP tasks. To tackle...

arxiv.org

Deployment pattern 2— Conversational AI

Traditionally, conversational flows have been highly scripted exchanges of “Bot says” -> “Human says” -> “Bot Says”...signifying different hypothetical real world scenarios also known as “Stories” for Rasa developers. Each user’s **intent** can be expressed as hundred of “stories” based on the user’s state and interactions, the bot takes actions to execute the defined story and respond back with canned responses. For e.g. If a user want to subscribe to a newsletter, there could be two paths:

1. User is already subscribed.
2. User is not subscribed.

source

If the user triggers the intent by saying “How do I subscribe to the newsletter”, the bot needs to check if the user is already subscribed or not and then take the appropriate next step. This decision making of “what to do next” is a manually hard-coded path. Any diversion from the path and the bot should say, “Sorry, I am still learning, I can help you with xyz..”.

The real cost of building and maintaining bots came from these stories. The reason to do something as tedious a pattern as above was to enable the bot can navigate diverse real-world scenarios and new path could be added in an organized manner. But, the path writer always had some “conditions to check”, “actions to execute” and “end goal” of the conversation in mind to build a script which executes with a purpose.

With LLMs, we try to automate this script writing or path planning using “Reasoning” and “Planning” capabilities of LLMs — see more about this here, with a strong human in the loop system.

Imagine you are a customer service agent, a user came to you with the same request, how do i subscribe to your service? How would you define what next steps you should take? Can it be completely open ended? Most probably not, but it can’t be as heavily scripted as above either to manage costs. What if I told you the following:

> Condition — if **email** exists, user can **subscribe**
>
> Tools — check_subscription, add_subscription

Then as a self-respecting human, you would be able to stitch stories like the one below in your head:

1. User wants to subscribe, based on the statement — “How do I subscribe?”
2. ***Ask*** user for email — “What is your email?”
3. If he gives a valid email, ***trigger*** tool — check_subscription
4. If user has not subscribed, ***trigger*** add_subscription
5. ***Respond*** with success or failure.

And this is what we want the LLM to do, produce “Plans” which it can refer to as blueprints and act upon at run-time. See more about this break up of planning and reasoning here.

> *Going back to module template, let’s see what a planner might look like:*

The planner above can use tools and conditions to build plans or stories, at design or runtime. Let's see a real example of this in research:

KnowAgent: Knowledge-Augmented Planning for LLM-Based Agents

Large Language Models (LLMs) have demonstrated great potential in complex reasoning tasks, yet they fall short when...

arxiv.org

KnowAgent: Knowledge-Augmented Planning for LLM-Based Agents

What are some tools which can help the planner at run-time to decide the path with reliable reasoning?

1. Previous paths triggered by similar statements.
2. **Enterprise graph of actions**, and dependency between actions. This will help the planner decide if one action would lead to the correct result, which can then lead to the next action and so on until recursively it solves the problem. See this for a related real world integration of knowledge graphs and planning with Neo4J and Langchain, not necessarily related to planning paths.
3. Current state of the user / conversation.

Deployment pattern 3 — Multi — Agent

In a multi-agent setting, the goal is to define roles and responsibilities for LLM powered generators with precise tools which can work together to come up with an intelligent answer / solution.

Due to the well defined roles and underlying models, each Agent can delegate a sub-goal or part of the "**Plan**" to the "Expert" and then use the output to decide what to

do next. See the GPTPilot example in the end for more details.

Patterns like these are executed using the following communication patterns which control the authority of defining next steps in the chain of execution. See Orchestrating Agentic Systems for more on this.

Benefits of Multi-Agent Designs:

- **Separation of concerns**: Each agent can have its own instructions and few-shot examples, powered by separate fine-tuned language models, and supported by various tools. Dividing tasks among agents leads to better results. Each agent can focus on a specific task rather than selecting from a multitude of tools.
- **Modularity**: Multi-agent designs allow breaking down complex problems into manageable units of work, targeted by specialized agents and language models. Multi-agent designs allow you to evaluate and improve each agent independently without disrupting the entire application. Grouping tools and responsibilities can lead to better outcomes. Agents are more likely to succeed when focused on specific tasks.
- **Diversity**: Bring in strong diversity in the agent-teams to bring in different perspectives and refine the output and avoid Hallucinations & Bias. (Like a typical human team).
- **Reusability**: Once the agents are built, there is an opportunity to reuse these agents for different use cases, and think of an ecosystem of agents, that can come together to solve the problem, with a proper choreography/orchestration framework (such as AutoGen, Crew.ai etc)

Deployment pattern 4— CoPilot

The only difference I see in a CoPilot system is the learning it gains by interacting with the user and test functions.

More to come..

Frameworks vs Implementations

It is important to differentiate between *frameworks* of building these CoPilots, and the actual *implementations* of CoPilots (like GPT Pilot and aider). In most scenarios, none of the open-source CoPilots have been developed on the frameworks, rather all implementations have been developed from scratch.

Popular implementations we will review: OpenDevin, GPT Pilot

Popular research papers we will review: AutoDev, AgentCoder

Popular frameworks — Fabric, LangGraph, DSPy, Crew AI, AutoGen, Meta GPT, Super AGI etc.

As much as possible, we will try to stick to the following definitions for LLM based Multi Agents

Deep Dive — apps

GPT Pilot

The GPT Pilot project, is a brilliant example of creative prompt engineering and chaining of LLM responses in an "layered" flow to execute tasks which seem complex.

There are several profiles which work in a layered communication manner, see the green boxes below:

https://www.reddit.com/r/MachineLearning/comments/165gqam/p_i_created_gpt_pilot_a_research_project_for_a/

The individual agents interact in a layered manner, where the agents are triggered from one node to the next, without any decision making agent in the picture.

There are some beautiful principles which the product has deployed which makes it work well:

1. Breaking tasks down into small achievable modules for the LLM to generate code for.
2. Test driven development, collect good use-cases from the human to be able to validate and iterate with accuracy.
3. Context rewinding and code summarization.

Despite all of the above complex prompt engineering and flow design, the benefits of fine-tuning for each of the agents can not be stressed less, to reduce costs and improve accuracy.

Agents | Llm | AI | Langgraph | Gpt

Follow

Written by Raunak Jain

575 followers · 546 following

Seeking patterns and building abstractions

Responses (18)

Bgerby

What are your thoughts?

Raunak Jain Author

Apr 15, 2024

Thank you guys! My first story ever, so happy to see it's helping and keeping people engaged

3 Reply

Gooseecloud

Apr 15, 2024

Thank you very much, in-depth article. Through LLM, do we tend to do AI-enhanced deterministic workflows, or LLM autonomous workflows? How do we choose the balance between the two? The latter gives us space but also brings a lot of uncertainty.

5 1 reply Reply

Steve Cain
Apr 15, 2024

What a thorough, deep dive into compound AI systems. I enjoyed it!

Here's a flowchart summarizing the key components and deployment patterns discussed in the article about compound AI systems:

![Compound AI Systems... more

7 Reply

See all responses

More from Raunak Jain

Raunak Jain

Cognitive AI Systems & LLMs

Driving human — like capabilities for self-learning, critiquing, problem-solving, planning, reasoning..

Apr 5, 2024 79

Raunak Jain

Orchestrating agentic systems

Static and dynamic execution of tasks by agents

Apr 14, 2024 202

Raunak Jain

Fine-tuning LLMs for Enterprise RAG — A design perspective

How to know if you are training for getting better or just burning money?

Apr 1, 2024 120

Raunak Jain

Path to production for LLM agents

Decoupling Reasoning & Planning from Action & Observations

Mar 15, 2024 136 1

See all from Raunak Jain

Recommended from Medium

In Towards AI by Teja Kusireddy

We Spent $47,000 Running AI Agents in Production. Here's What Nobody Tells You About A2A and MCP.

Multi-agent systems are the future. Agent-to-Agent (A2A) communication and Anthropic's Model Context Protocol (MCP) are revolutionary. But...

Oct 16 2.8K 91

In Data Science Collective by Ida Silfverskiöld

Agentic AI: Single vs Multi-Agent Systems

Building with a structured data source in LangGraph

Oct 28 645 12

In UX Collective by Iasonas Georgiadis

Building AI-driven workflows powered by Claude Code and other tools

How agentic CLI tools extend Figma MCP and turn wireframes into production-ready prototypes

Oct 29 277 1

Taner Tombaş

Intent Detection for AI Systems: Understanding What Users Really Want

How to build intelligent query understanding that goes beyond keyword matching

Jun 17 30

In MongoDB by MongoDB

Build AI Agents Worth Keeping: The Canvas Framework

This article was written by Mikiko Bazeley.

Oct 8 110 5

Tapobrata Chatterjee

GenAI—Evolution from LLM to Agentic AI

The world of tech—and honestly, beyond tech too—has been shaken up by this “dangerous” yet fascinating subject called GenAI. The level...

Jul 22 12 1

See more recommendations