LLM Agents: Implementations

Yashila Bordag

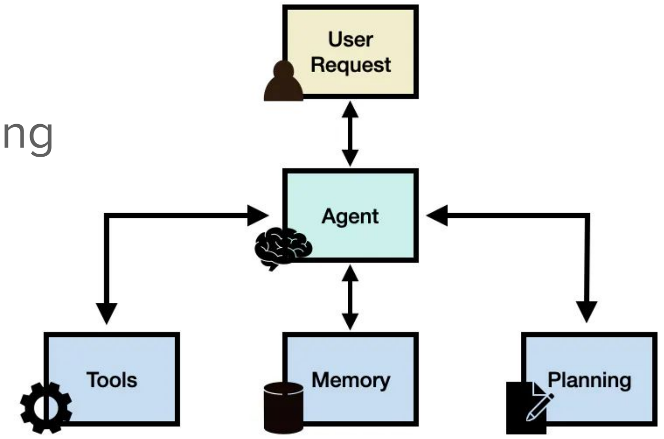
Recap: What are LLM Agents?

Agent - “An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.”

- Goal-based Agents
- Utility-based Agents

LLM Agent - instance of a structured prompting system which recursively calls an underlying LLM instance to execute complex tasks

LLM Agents perform significantly better than simple LLM prompting, using tools, memory, and planning to overcome LLM limitations



Step-by-Step guide to making well-defined agents

1. Define your Task
2. Break out Static Sub-Processes and define Tools
3. Define Agents
4. Choose your LLM
5. Choose your Agentic Framework
6. Use an LLM to help you generate prompts for Agent Framework based on steps 1-3
7. Implement Agents

Decide on your Task

LLMs work best when they have well defined prompts. Well defined goals are necessary for the Agent to understand the task is complete

- Define your goals, what actions need to be done, and what subjective standards do you have the output? Describe what a completed task should produce or look like.
- Clearly define your expected inputs and outputs, especially if you require structured outputs like JSON data
- You can roughly describe an example workflow for the agent to use based on your experience
- Try to break out parallelizable subtasks from this task which the agent can assign to subagents. For each subtask, repeat this step as necessary.

This will help you define your task prompt which roughly corresponds with your agentic ‘goal’ by the Norvig-Russel definition

How to Approach Defining Tools

General Rule of Thumb: save the reasoning capacity of the LLM for what is best solved by using dynamic LLM reasoning

- Non-verbal reasoning tasks like making calculations, low-level data processing, and simple logical reasoning which can be implemented easily in code should be abstracted into function tools
- Extremely simple verbal tasks like generating labels or doing surface level editing can be turned into functions using LLMs through langchain.

Defining your Agents

Once you have your Tasks and Tools, consider what attributes and behaviors would best complete the task while using the tools available

Common practices:

- Define agent attributes similarly to job description
- Describe the agent as ideally as possible

This will help you define your agent prompt which tells your agents which ideal behavior or ‘heuristic’ to follow, roughly corresponding with your agentic ‘utility’ by the Norvig-Russel definition

Deciding on which LLM to use

Big LLMs

- Pros: have more reasoning capacity, greater ability to interpret and execute on complex and ambiguous tasks, and can deal with more uncertainty during execution time, less likely to hallucinate, much larger prompt window
- Cons: very expensive, more computationally intensive, very large models not available locally, more likely to give an ambiguous answer

Small LLMs

- Pros: cheap, available locally, so computationally cheap it could run on a laptop, returns succinct straightforward answers
- Cons: require very thoroughly defined tasks and lots of prompt engineering/management, less flexible when encountering issues, more likely to hallucinate, much smaller prompt window

Deciding on which LLM to use

Your cost per task execution is pretty straight-forward:

- Cost for task = Cost per LLM call * number of times called
    - AWS pricing:
        - Claude 3.5 Opus - $0.015 per 1000 tokens in (prompt length), $0.075 per 1000 tokens out
        - Llama 3.1 8B - $0.0003 per 1000 tokens in, $0.0006 per 1000 tokens out
        - Llama 3.1 8B is 50x cheaper for tokens in, and 125x cheaper for tokens out, and still has fairly good performance

Running Locally with libraries like Ollama is also a popular solution, as you still have access to opensource models like Meta’s Llama, Mistral, Microsoft Phi, and Google Gemma

If your task is well defined and not very conceptually complicated, opt for a smaller model (~7B or 8B), otherwise scale up to a 70B or more as needed

If you are doing a proof of concept, use an 8B model locally to reduce cost (requires 16+ GB RAM)

Choosing a Framework

Langchain - good for fixed processes, and very simple agents, documentation can be difficult to read

CrewAI - built on langchain, very simple syntax and great examples, good for Proof of Concepts, but has issues with agentic reasoning (resulting in tons of calls or agents stuck in reasoning loops) and component flexibility

AutoGen - More structured library with more component flexibility, but the resulting code is much more complicated

LlamaIndex - Implementations of experimental agentic structures, and exposes functional parts of the agent for fine-grained control over behavior

LangGraph - from same group as LangChain, aims to provide heavy-duty abstract agents

Provider Agents - many service providers are giving users the ability to set up simple agents on their platforms, esp. for RAG, data processing, etc.; should be much simpler to set up than using full framework

Choosing a Framework (cont.)

As with choosing an LLM, choosing a framework requires some deliberation:

- If you are making a simple agent, you might have a lot of luck with LangChain, Provider Agents, or CrewAI
- I personally like CrewAI the most for simple POCs and have been using it in hackathons because of how straightforward the code and prompts are to explain to others
- Autogen seems to be the go to for more complicated projects, and LangGraph seems to have features intended for distributed production frameworks

All of these frameworks are in early development, and most Agent-base projects are still in the POC phase, new frameworks might be released which provide much better support for multi-agent, distributed production environment, and abstract reasoning use cases

Iterating on Your Prompts and Definitions with an LLM

Depending on your framework and the LLM you have chosen, you will have to come up with quite a few prompts to pass into your LLM through your code

Look through your framework's documentation to find out what prompts you need for your agents and tasks

Smaller LLMs work better with well defined prompts

- Need to be precise and concise, be clear, structured, and consistent
- Using LLMs will give you consistently good style, precision and formatting for prompts

You can also generate test cases, output structure, output examples for prompts, function descriptions for your tools, etc.

Tips on Developing Agents

Once you have your LLM, Framework, and prompts defined, implementing should be more straightforward

Prototype your agentic workflow with toy examples first, without tools, RAG, or structured outputs

- this sanity checks that the LLM you have chosen is able to reason appropriately about the task
- You can then add these pieces one by one and check if the agent still works well
- Always validate your outputs, this allows you to checkpoint whether the agent is working or not

Tips on Developing Agents (cont.)

LLMs are great tools for prompting, coding, making test examples, and documenting your progress

Roughly, we can also think about the prompts in terms of setting heuristics and goals

- Agent Prompt = Utility/Heuristic -> “Are you acting the way you should be?”
- Task Prompt = Goal -> “Have you achieved what you want to?”

The importance of this process is that this is a very new technology and some agentic workflows will be impossible to implement without enough critical consideration

LLM Agent Stack

- LLM
  - Local - Ollama (Workstations) / LightLLM (servers)
  - API Endpoints - OpenAI, Anthropic, AWS, Azure, Nvidia NIMs
- Python
- Python LLM Libraries
  - Langchain - defining functions and tools
  - LlamaIndex - defining indexes for RAG, other tooling functionality)
  - Pydantic/Instructor - defining structured outputs
- Agentic Framework
  - LangChain
  - CrewAI
  - AutoGen
  - LlamaIndex
  - LangGraph
  - Single Agent Systems (MetaGPT, MemGPT)
  - Provider Agents

Code Demo - Very Basic Knowledge Graph Extractor

https://github.com/ybordag/agents-crewai-demo

More Examples for Knowledge Graph Extraction

https://github.com/microsoft/graphrag

https://docs.llamaindex.ai/en/latest/examples/cookbooks/GraphRAG_v1/#graphragextractor

https://github.com/karthik-codex/Autogen_GraphRAG_Ollama