Keynote

Open source LLM: Llama, Mistral AI, Hugging Face, Cohere

Closed source LLM: OpenAI gpt

Open source framework for AI agentic flow: crewAI, Nexusflow, etc

AI agentic flow

- breakdown task into smaller one for agents to execute
- define agents to optimize prompting experience
- task by human > agent with tools, processes > answer by LLM

*tools: SKU/ capabilities
*processes: in series, in parallel

Multi AI agent system

- multi agents
- each agent is independent, can run from diffrenet LLMs, i.e. agent A: use GPT, agent B: use Llama
- each agent focuses to execute its task only
- each task is unique
- a task has >= 1 agents, i.e. collection of agents = agent crew
- agent crew auotmate the process themselves
- e.g.1. write a reserach paper, need 3 agents, i.e. researcher, writer, fact checker
- e.g.2. build a website, need 3 agents, i.e. web designer, software engineer, testing engineer
- e.g.3. data collection and analysis for sale team, need 3 agents, i.e. data collecter, data analyzer, scoring
- e.g.4. build a resume for engineer, need agents, i.e. job searcher, profiler, resume strategist, interview preparer
- task: search internet, read job websites, read resume, perform RAG on resume

Key requirement

- well understand your task goal and what real-world roles exist to deal with your task, this is helpful to define in detail

Outline for building AI agentic flow

- Goal
- Process, i.e. step 1 to step n

AI agent system use cases

- provide sales pitch
- provide marketing strategy
- research, writing and publishing
- financial analysis

Building blocks/ principles for a good agent

- role playing
- focus
- tool
- collaboration
- guardrail
- memory

*also a block for good human employees

Role playing
- each agent has a specific tittle, context that yields for specific response

Focus
- each agent focus to execute its own task

Tool
- each agent has specific tools, not too many to avoid task confusion
- 3 properties: versatile, fault-tolerant, caching
- versatile: multi-function, can manage all types of input and respond with strong output
- fault-tolerant: continue to execute and send message back to agent for correction, no stop
- caching: store the same previous task request, prevent same requests from different agents, avoid hit API rate limit, save time to execute, i.e. cross caching layer

Collaboration
- agents talk to each other to collaborate, delegate tasks, execute the main task as a whole

Guardrail
- default in crewAI framework, prevent AI hallucination
- ensure agents achieve what they should do

Memory
- 3 types: short term, long term, entity memory
- short term memory: within crew execution
- long term memory: after crew execution, self-improved and reuse
- entity memory: short term, within crew execution, by category, i.e. person name, org name, etc

3 types of AI agentic process

- in parallel, i.e. all agents execute their own tasks at the same time
- in series/ sequential, i.e. each agent execute its own task one by one before going to next agent
- in hierarchical, i.e. crew manager agent delegate tasks to different agents to execute

*crew manager agent: defined by human
*hierarchical + series/ parallel can happen at the same time
*crew manager agent and other agents can delegate tasks among themselves

Traditional software development

- absolute input > absolute transofmation > absolute output
- one system serve one purpose

AI software development, e.g. ChatGPT

- fuzzy input > fuzzy transofmation > fuzzy output
- fuzzy = various possibilities
- one system serve various purposes

Difference between prompting and crewAI

- crewAI allows multi agents ; other common prompting only has one centralized agent
- crewAI task description uses """ for each new line ; common prompt uses \ for each new line
- crewAI backstory = common prompt's prompt
- crewAI breakdowns each step to help define the agent and task ; no key required info and framework outlined in common prompt
- crewAI is fault-tolerant and prevent stop execution ; other frameworks stop execution when it comes to error

Key libraries

crewAI
- Agent
- Task
- Crew
- Process

crewai_tools
- SerperDevTool
- ScrapeWebsiteTool
- WebsiteSearchTool
- BaseTool

pydantic
- BaseModel

crewAI AI agent template - key

Step 1: Install crewAI
```
!pip install crewai==0.28.8 crewai_tools==0.1.6 langchain_community==0.0.29
```

Step 2: Control warning
```
import warnings
warnings.filterwarnings('ignore')
```

Step 3: Import crewAI
- from crewai import Agent, Task, Crew, Process

Step 4: Import LLM for building agents, i.e. openai, llama
```
import os
from utils import get_openai_api_key
openai_api_key = get_openai_api_key()
os.environ["OPENAI_MODEL_NAME"] = 'gpt-3.5-turbo'
```

Step 5a: Define Agent level tool (if needed)

Step 5: Define Agent, i.e. role, goal, backstory
```
agent_name = Agent(
    role=" ",
    goal=" ",
    tool=[ ],
    backstory=" ",
    allow_delegation=False,
    verbose=True
)
```

Step 6a: Define Task level tool (if needed)

Step 6: Define Task, i.e. description, expected output, related agent
```
task_name = Task(
    description=" ",
    tool=[ ]
    expected_output=" "
    context=[ ]
    human_input=True,
    async_execution=True,
    output_json=,
    output_file=" ",
    agent=" "
)
```

Step 7: Define Crew, i.e. connect Agent and Task together
```
crew = Crew(
    agents=[1,2,3],
    tasks=[1,2,3],
    process=Process.hierarchical,
    verbose=2,
    memory=True
)
```

Step 8: Run Crew
```
result = crew.kickoff (inputs={" "})
```

crewAI AI agent template - add tool if needed

.

Step 5a/ 6a: Define Tool (crewAI)

```
from crewai_tools import SerperDevTool, \
                         ScrapeWebsiteTool, \
                         WebsiteSearchTool

search_tool = SerperDevTool()
scrape_tool = ScrapeWebsiteTool()

OR ScrapeWebsiteTool(
    website_url="https://docs.crewai.com/how-to/Creating-a-Crew-and-kick-it-off/"
)
```

Step 5a/ 6a: Define Tool (custom)

e.g. build sentiment analysis tool, need to define name and description of custom tool everytime

```
from crewai_tools import BaseTool

class SentimentAnalysisTool(BaseTool):
    name: str ="Sentiment Analysis Tool"
    description: str = ("Analyzes the sentiment of text "
         "to ensure positive and engaging communication.")

    def _run(self, text: str) -> str: # Your custom code tool goes here
        return "positive"

sentiment_analysis_tool = SentimentAnalysisTool()
```

Create pydantic object as a class, agent can update the information of the class when having different variables

e.g. build a venue detail class and ask agent to fill in info below with different venues

```
from pydantic import BaseModel

class VenueDetails(BaseModel):
    name: str
    address: str
    capacity: int
    booking_status: str
```

*Pydantic: crewAI library

Template remarks

*agent: can be >=1, recommend to define QA agent in every scenario to check our agent work, i.e. QA can delegate tasks
*no. of task = no. of agent, i.e. suppose each agent has its one task
*agent backstory, task description: similar to prompt in prompting
*clear and specific role: define with specific title and level, e.g. JP Morgan senior financial analyst
*clear and specific backstory, description: list out required output amount, format, i.e. markdown, JSON, HTML, language, word limit, tone
*verbose = 2: see all the logs of the execution, range between 1 and 2 and True
*run crew input: define with variables, keywords set up in backstory (Agent) and description (Task), can set up separately before adding kickoff
*memory: define inside Crew, true means to include all 3 memories
*tool: define before Agent/ Task, depend on use case
*output_file: json, html, md (markdown)
*human_input: need human feedback or not
*async_execution: decide if in series/ in parallel to execute, depend on use cases
*process: define inside Crew, i.e. process=Process.hierarchical
*when in hierarchical, no need to define the agent and task order in Crew, let manager agent do it
*context: define task by referencing previous tasks

Use crewAI tool after defining agent/ task

- before agent: agent level tool, agent can use it in any of its tasks
- before task: task level tool, agent ONLY use it when executing that task

*task level tool (specific) overrides agent level tool (general)

Must use second person perspective (You) to define role, task, i.e. you are an analyst and need to analyze...