Introduction

Sajeed Ahmed

Chat interface : https://huggingface.co/spaces/asajeed/whoissajeed

Ask questions

- Describe Sajeed in one sentence?
- Describe Sajeed personal life?
- Where Sajeed worked?
- Describe Sajeed strengths?

Demo: Resume AI Assistant

Intelligent chatbot that answers questions about you using your resume

Resume Processing

Loads resume as knowledge base

Hugging Face Space

Deploy with Gradio, zero infrastructure

LLM Integration

Connects to Claude, DeepSeek, or GPT

Interactive Chat

Web interface for recruiters to learn about you

Smart Notifications

Pushover alerts for unanswerable questions

Privacy Control

Get alerts for out-of-scope questions

Implementation Architecture

1. Setup & Configuration

```
import gradio, litellm, requests
```

Load resume, configure LLM and Pushover

2. Question Analysis

```
Check if answer exists in resume
```

LLM determines information availability

3. Response Generation

```
completion(model, messages)
```

Generate contextual answer from resume

4. Notification Trigger

```
send_pushover(question)
```

Alert via Pushover for unknown questions

Deployment

```
gr.Interface(chat_fn, inputs, outputs).launch()
```

CrewAI

Multi-Agent Framework for Production AI Systems

Building Collaborative AI Teams with Python

Technical Deep Dive • 2026

Agenda

1. Framework Overview

What is CrewAI and why it matters

2. Core Concepts

Agents, Tasks, and Crews explained

3. Development Setup

Using Cursor IDE and uv package manager

4. Live Demo

Building an Engineering Team crew

5. LLM Integration

Configuring Claude and DeepSeek models

6. Best Practices

Production tips and patterns

What is CrewAI?

A Python framework for orchestrating role-playing autonomous AI agents to work together intelligently on complex tasks.

🤖 Role-Based Agents

Each agent has specific expertise, goals, and backstory for specialized task execution

🔄 Collaborative Workflow

Agents work together through sequential or hierarchical processes to solve complex problems

⚡ Production Ready

Built-in memory, tools integration, and support for multiple LLM providers

Key Insight: CrewAI enables you to build AI systems that mirror human team structures with specialized roles and collaborative workflows.

Core Concepts

Agents

Autonomous units with defined roles and capabilities

```
role, goal, backstory, llm, tools
```

Crew

Orchestrates agents and tasks to achieve complex goals

```
agents, tasks, process, verbose
```

Tasks

Specific assignments with clear objectives and expected output

```
description, expected_output, agent
```

Process

Execution workflow for task completion

```
Sequential | Hierarchical
```

Development Setup

Cursor IDE

AI-powered code editor built on VSCode

- Intelligent code completion
- AI chat within your codebase
- Visit https://www.cursor.com/
- Download and follow its instructions to install and open Cursor

uv Package Manager

Fast Python package installer

- 10-100x faster than pip
- Built in Rust
- https://docs.astral.sh/uv/getting-started/installation/
- uv self update,

Quick Start

uv sync , uv init my-project

uv pip install crewai litellm or uv tool install crewai or uv tool upgrade crewai

Create your API key, OpenAI key is at https://platform.openai.com/api-keys , Claude API at https://console.anthropic.com/ from Anthropic
OPENAI_API_KEY=xxxx, GOOGLE_API_KEY=xxxx ANTHROPIC_API_KEY=xxxx DEEPSEEK_API_KEY=xxxx

Five Step CrewAI

1. Create the project

```
crewai create crew my_project
```

2. Fill in the config yaml files

Define Agents and Tasks in configuration files

3. Complete the crew.py module

Create Agents, Tasks, and Crew referencing config

4. Update main.py

Set configuration and run the crew

5. Run with command: crewai run

Demo: Engineering Team Crew

Building a collaborative AI engineering team to design, develop, and test software

Engineering Lead

Creates technical design and architecture

```
Task: design_task
```

Frontend Engineer

Builds user interfaces and experiences

```
Task: frontend_task
```

Backend Engineer

Implements server-side logic and APIs

```
Task: backend_task
```

Test Engineer

Validates quality and functionality

```
Task: test_task
```

Live Demo

Agent Configuration (agents.yaml)

```yaml
engineering_lead:
  role: >
    Engineering Lead for the engineering team, directing the work of the engineer
  goal: >
    Take the high level requirements described here and prepare a detailed design for the
backend developer;
    everything should be in 1 python module; describe the function and method signatures in
the module.
    The python module must be completely self-contained, and ready so that it can be tested
or have a simple UI built for it.
    Here are the requirements: {requirements}
    The module should be named {module_name} and the class should be named {class_name}
  backstory: >
    You're a seasoned engineering lead with a knack for writing clear and concise designs.
  llm: deepseek/deepseek-coder

backend_engineer:
  role: >
    Python Engineer who can write code to achieve the design described by the engineering
lead
  goal: >
    Write a python module that implements the design described by the engineering lead, in
```

Task Configuration (tasks.yaml)

```yaml
design_task:
  description: >
    Take the high level requirements described here and prepare a detailed design for the
engineer;
    everything should be in 1 python module, but outline the classes and methods in the
module.
    Here are the requirements: {requirements}
    IMPORTANT: Only output the design in markdown format, laying out in detail the classes
and functions in the module, describing the functionality.
  expected_output: >
    A detailed design for the engineer, identifying the classes and functions in the
module.
  agent: engineering_lead
  output_file: output/{module_name}_design.md

code_task:
  description: >
    Write a python module that implements the design described by the engineering lead, in
order to achieve the requirements.
    Here are the requirements: {requirements}
  expected_output: >
    A python module that implements the design and achieves the requirements.
```

Crew Implementation (crew.py)

```python
from crewai import Agent, Crew, Process, Task
from crewai.project import CrewBase, agent, crew, task
@CrewBase
class EngineeringTeam():
    """EngineeringTeam crew"""

    agents_config = 'config/agents.yaml'
    tasks_config = 'config/tasks.yaml'

    @agent
    def engineering_lead(self) -> Agent:
        return Agent(
            config=self.agents_config['engineering_lead'],
            verbose=True,
        )

    @agent
    def backend_engineer(self) -> Agent:
        return Agent(
            config=self.agents_config['backend_engineer'],
            verbose=True,
            allow_code_execution=True,
```

LLM Integration

CrewAI supports multiple LLM providers through LiteLLM

Claude (Anthropic)

```
ANTHROPIC_API_KEY=sk-ant-...
llm: anthropic/claude-3-7-sonnet-latest
```

Most capable model with strong reasoning

DeepSeek

```
DEEPSEEK_API_KEY=sk-...
llm: deepseek/deepseek-chat
```

Cost-effective and fast ($0.14/M tokens)

Install: uv pip install litellm anthropic

Process Types

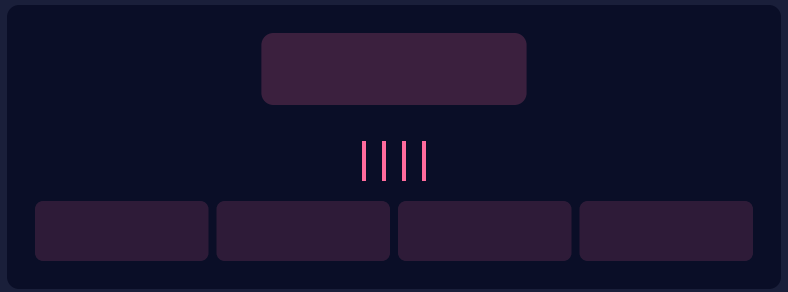
Sequential Process

Tasks execute one after another in a defined order. Each task can use outputs from previous tasks as context.

```
process=Process.Sequential
```

Hierarchical Process

A manager agent coordinates and delegates tasks to other agents. Ideal for complex workflows requiring dynamic task allocation.



```
process=Process.Hierarchical
```

Best Practices

Clear Role Definition

Give each agent a specific, well-defined role with clear expertise boundaries

Detailed Backstories

Rich backstories help agents understand their expertise and approach tasks authentically

Task Context

Use context to pass outputs between tasks for informed decision-making

Expected Output

Define precise expected outputs to guide agents toward desired results

Enable Verbose Mode

During development, use verbose=True to debug agent reasoning and task execution

Memory Management

Leverage agent memory for context retention across multiple executions

Advanced Features

🛠️ Tool Integration

Equip agents with external tools for enhanced capabilities

```
File operations, web search, API calls, code execution
```

📊 Output Validation

Pydantic models for structured, validated outputs

```
Type safety, automatic validation
```

🧠 Memory Systems

Three types of memory for context retention

```
Short-term, Long-term, Entity memory
```

🔌 Custom Tools

Build specialized tools for domain-specific needs

```
@tool decorator, custom implementations
```

🔄 Callbacks & Monitoring

Track execution progress and agent decisions

```
Real-time logging, performance metrics
```

⚙️ Delegation

Agents can delegate subtasks to other agents

```
allow_delegation=True
```

Pros and Cons of CrewAI

✅ Advantages

Role-based Design
Agents have specific roles and expertise

Collaborative Intelligence
Agents work together and delegate tasks

Flexible Architecture
Easy to customize and extend

Process Control
Define how agents interact and hand off work

Tool Integration
Seamlessly connect to external APIs

⚠️ Considerations

Resource Usage
Multiple agents consume more resources

Coordination Overhead
Agent communication adds latency

Debugging Challenges
Harder to trace issues across agents

Cost Management
Multiple LLM calls can be expensive

Trade-offs

Ready to Build?

Key Takeaways

✓ CrewAI enables building collaborative AI teams with specialized roles

✓ Sequential and Hierarchical processes for different workflow needs

✓ Flexible LLM integration with Claude, DeepSeek, and others

✓ Modern tooling with Cursor IDE and uv package manager

docs.crewai.com

💬 **github.com/joaomdmoura/crewAI**

Thank you!