

[Open in app](#) ↗

Medium

 Search

★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



Step-by-Step: Building Intelligent Agents Using crewai, Groq or Ollama in Your Local System. (With Project)

Samar Singh · [Follow](#)

7 min read · Apr 25, 2024



Listen



Share



More

Imagine crafting an intelligent agent that can hold conversations, access information, and even utilize various tools at your command. This isn't science fiction — it's achievable on your local system! This guide will walk you through the steps of building your own intelligent agent.

Before explaining How to build and run AI agents locally without relying on external APIs. Let me Explain what are Agents.

AI agents, also known as intelligent agents, are software entities that perceive their environment, make decisions, and take actions to achieve specific goals. These agents are equipped with AI capabilities, such as natural language processing, different tools allowing them to perform tasks autonomously or assist humans in various tasks.

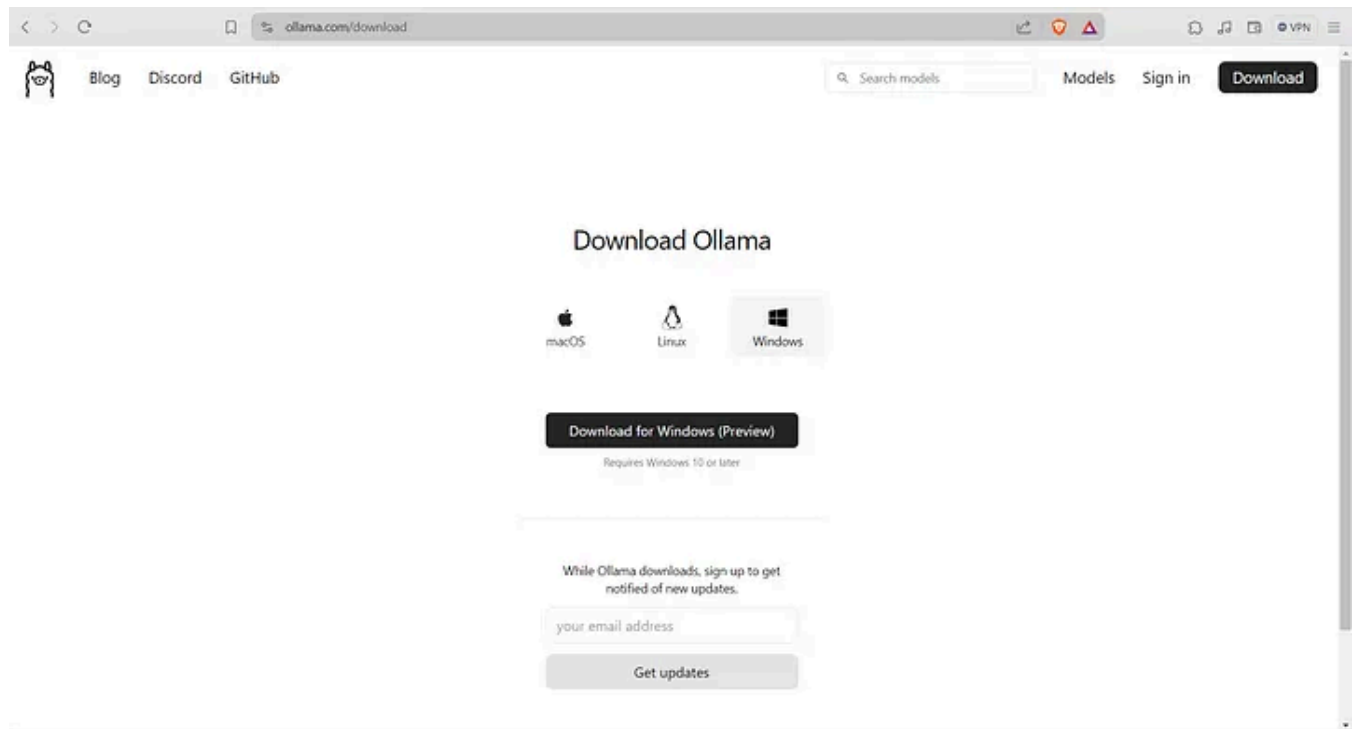
Imagine this: No more endless scrolling through webpages for research! With AI agents, we're revolutionizing the way we find information. You just type in what you need, and bam! These smart tools dive into the web for you, gathering all the important bits from different sites. What you get? A neat summary, saving you time and hassle. It's like having your own personal researcher at your fingertips!

Building AI Agents Locally: While AI agents offer tremendous potential, concerns over privacy and reliance on external APIs have prompted the development of tools

for building and running AI locally. Ollama and Langchain and crewai are such tools that enable users to create and Use AI agents on their own hardware, keeping data private and reducing dependency on external services.

Lets start with installation. How to install or run the llama locally.

For that we need to go to [ollama](https://ollama.com) site. The website will look like this.



Based on your Operating system. You can download the ollama which will allows us to run the LLM models locally. If you want to see all available models you can click on Models on the upper right side.

```
# Run this command in terminal. It will take time to download the model locally
ollama run llama3
# Similarly any other model if you want to download you just need to type the
## model name after ollama run.
ollama run mistral
# To test the model is working type any prompt after the running above command.
## like "tell me a Joke"
```

That was the installation and running ollama locally. Now lets jump to our project. In which i will explain how to build AI Agents with locall LLM. I will also explain how to use LLM from Groq which is quite faster and free.

First Let me explain The project which is **Customer Support Ticket Classification and Action**. In this project we will develop a system that automatically classifies incoming customer support tickets into relevant categories and takes appropriate actions based on the classification. The system should be able to distinguish between urgent issues requiring immediate attention, routine inquiries, and non-support-related messages. **Upon classification, the system should take one of the following actions:**

Urgent Issue: Tickets classified as urgent should trigger an immediate notification to the appropriate support personnel or department and escalate for immediate resolution.

Routine Inquiry: Tickets classified as routine inquiries should be assigned to the relevant support team for response within the standard response time.

Non-Support-related Message: Tickets classified as non-support-related messages (such as feedback, general inquiries, or spam) should be automatically filtered out and not responded to. Below is the project screenshot.

```

File Edit Selection View Go Run Terminal Help
AI Agent
app.py local_llm_ollama.py A
app.py > ...
7 ticket = "Customer support ticket: Unable to access account"
9

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

(al.env) C:\Users\samar\Pycharm\Projects\AI Agent>python app.py
[DEBUG]: == Working Agent: Ticket Classifier
[INFO]: == Starting Task: Classify the following support ticket: 'Customer support ticket: Unable to access account'

> Entering new CrewAgentExecutor chain...
I now can give a great answer.

Final Answer: urgent

> Finished chain.
[DEBUG]: == [Ticket Classifier] Task output: urgent

[DEBUG]: == Working Agent: Ticket Action Taker
[INFO]: == Starting Task: Take appropriate action on the support ticket: 'Customer support ticket: Unable to access account' based on the classification provided by the 'classifier' agent.

> Entering new CrewAgentExecutor chain...
I now can give a great answer.

Final Answer:
Urgent: Unable to Access Account - Priority 1

Action Taken:
Notify Urgent Issue to Tier 1 Support Team Lead via Email and SMS.
Assign Ticket to Tier 1 Support Team for Immediate Resolution.
Add Note to Ticket: "Urgent: Customer unable to access account. Please resolve ASAP."
Escalate Ticket to Senior Support Agent if not resolved within 30 minutes.
CC: Senior Support Agent, Support Team Lead on the ticket for awareness.
Status Update: In Progress
  
```

Now lets Jump to the code.

```
# First Lets install packages
pip install crewai, langchain_community
```

```
# Now import these libraries
from crewai import Agent, Task, Crew, Process
from langchain_community.llms import ollama
```

Crewai is a framework designed to create and manage teams of AI agents working together. Think of it like a platform for building a well-oiled crew, but instead of human workers, you have AI agents with different specialties.

LangChain is a framework designed to simplify the process of building applications that use large language models (LLMs). LangChain is an open-source. LangChain provides tools and abstractions to improve the customization, accuracy, and relevancy of the information the models generate. For example, developers can use LangChain components to build new prompt chains or customize existing templates. Here we are using to load the Ollama.

1. Defining the Agents:

```
model = ollama(model="llama3")

ticket = "Customer support ticket: Unable to access account"

# Define the Ticket Classifier Agent
# This defines the ticket classifier agent. It's responsible for classifying incoming tickets
classifier = Agent(
    role="Ticket Classifier",
    goal="Accurately classify incoming customer support tickets into relevant categories",
    backstory="You are an AI assistant tasked with classifying incoming customer support tickets",
    verbose=True,
    allow_delegation=False,
    llm= model
)

# Define the Ticket Action Agent
# This defines the ticket action taker agent. It's responsible for taking appropriate actions based on the classification
action_taker = Agent(
    role="Ticket Action Taker",
    goal="Take appropriate actions based on the classification of the customer support ticket",
    backstory="You are an AI assistant responsible for taking actions on classified tickets",
    verbose=True,
    allow_delegation=False,
```

```
llm= model  
)
```

The agents are responsible for classifying customer support tickets and taking appropriate actions based on the classification.

2. Defining Tasks:

```
# This task is to classify the support ticket using the classifier agent.  
classify_ticket = Task(  
    description=f"Classify the following support ticket: '{ticket}'",  
    agent=classifier,  
    expected_output="One of these three options: 'urgent', 'routine', or 'non-s"  
)  
  
# This task is to take appropriate action on the support ticket based on the cl  
take_action_on_ticket = Task(  
    description=f"Take appropriate action on the support ticket: '{ticket}' bas  
    agent=action_taker,  
    expected_output="An action taken on the ticket based on the classification"  
)
```

3. Creating the Crew: Here, we create a crew with the classifier and action taker agents, and the tasks defined earlier. We set the verbosity level to 2 for detailed output and specify the process to be sequential, meaning tasks will be executed one after the other.

```
crew = Crew(  
    agents=[classifier, action_taker],  
    tasks=[classify_ticket, take_action_on_ticket],  
    verbose=2,  
    process=Process.sequential  
)
```

4. Executing the Crew:

```
output = crew.kickoff()  
print(output)
```

Now lets do with groq. But why should i use groq?. Running the model locally is secure but slow if you have GPU than it will run fine in case of CPU its quite slow. So thats where groq comes. Everything will remain same in code just need to do small changes as below. But first visit groq.com and login its free and fast. Than click on GroqCloud which is in center below.

Now click on API Keys and create a new api keys.

Now copy the Groq-api-key and paste in below code. "groq-api-key". You are ready to run your code. Remember we are not using Ollama in below code so remove the model variable from both agents or you can copy paste below code.

```
from crewai import Agent, Task, Crew, Process
import os

os.environ["OPENAI_API_BASE"] = "https://api.groq.com/openai/v1"
os.environ["OPENAI_MODEL_NAME"] = "llama3-70b-8192"
os.environ["OPENAI_API_KEY"] = "groq-api-key" # paste your api key here

ticket = "Customer support ticket: Unable to access account"

# Define the Ticket Classifier Agent
classifier = Agent(
    role="Ticket Classifier",
    goal="Accurately classify incoming customer support tickets into relevant categories",
    backstory="You are an AI assistant tasked with classifying incoming customer support tickets",
    verbose=True,
    allow_delegation=False
)

# Define the Ticket Action Agent
action_taker = Agent(
    role="Ticket Action Taker",
    goal="Take appropriate actions based on the classification of the customer support ticket",
    backstory="You are an AI assistant responsible for taking actions on classified customer support tickets",
    verbose=True,
    allow_delegation=False
)

# Define the Task: Classify the Ticket
classify_ticket = Task(
    description=f"Classify the following support ticket: '{ticket}'",
    agent=classifier,
    expected_output="One of these three options: 'urgent', 'routine', or 'non-urgent'"
)

# Define the Task: Take Action on the Ticket
take_action_on_ticket = Task(
    description=f"Take appropriate action on the support ticket: '{ticket}' based on classification",
    agent=action_taker,
    expected_output="An action taken on the ticket based on the classification"
)

# Create the Crew
crew = Crew(
    agents=[classifier, action_taker],
    tasks=[classify_ticket, take_action_on_ticket],
    verbose=2,
```

```
process=Process.sequential
)

# Execute the Crew
output = crew.kickoff()
print(output)
```

Here is the Git Hub link of the project. [GithuLink](#)

“Finally, it’s completed. I hope you enjoyed it and learned something new today. Let me know if you have any doubts or suggestions.”

If you are interested in How to build RAG based project. I have written a Article on it. [A Deep Dive into Retrieval-Augmented Generation \(with Project\)](#).

AI

Ai Agent

Llm

Ollama

Groq



Follow

Written by Samar Singh

457 Followers · 13 Following

I Build Scalable AI Solutions 🚀 | Empowering Businesses with Intelligent Automation. Let's connect:-
<https://www.linkedin.com/in/samar-singh-119855206>

Responses (1)



What are your thoughts?

Respond



Sirine Jnayeh

about 1 month ago



Thank you for well documented article , but you didnt mention your LLM provider .



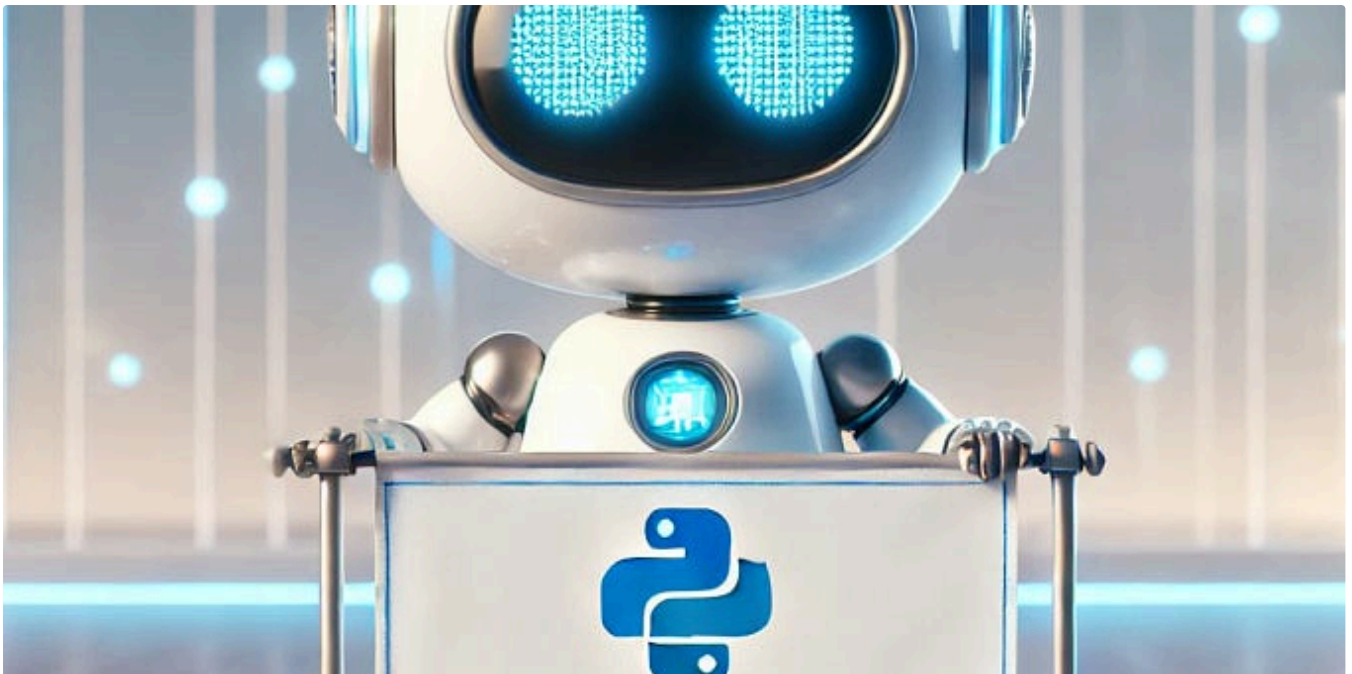
1




1 reply

Reply

More from Samar Singh

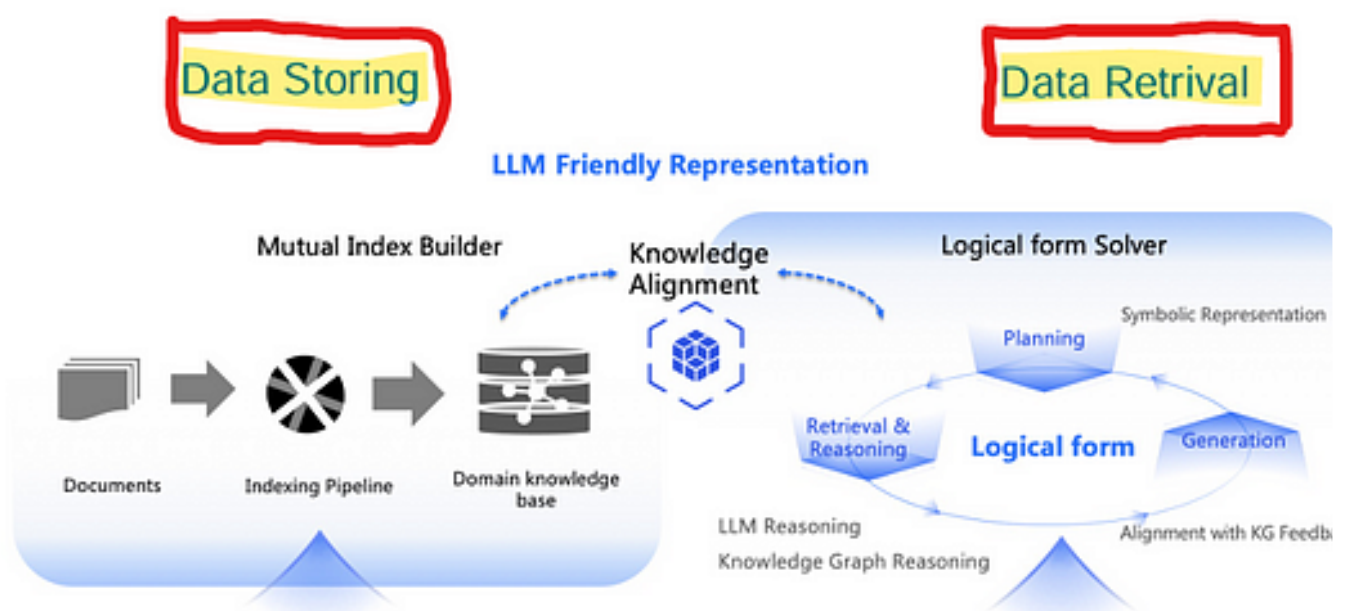


 Samar Singh

Pydantic AI: The Python Agent Framework to BUILD Production-Grade AI Agents!

Pydantic, a powerhouse in the Python ecosystem with over 285 million monthly downloads, has been a cornerstone of robust data validation in...

★ Dec 8, 2024 🖱️ 190 💬 3



 Samar Singh

Why Knowledge Augmented Generation (KAG) is the Best Approach to RAG

In the ever-evolving landscape of artificial intelligence, the introduction of Knowledge Augmented Generation (KAG) marks a pivotal...



Dec 29, 2024



279



2



Samar Singh

Ollama Introduces Structured Outputs: A Leap Towards Precision and Consistency

Ollama, a pioneering platform in the AI space, has unveiled a transformative feature—structured outputs. This innovation allows users to...



Dec 10, 2024



126





Samar Singh

LightRAG : A GraphRAG Alternative.

How to Set Up LightRAG Locally?



Oct 25, 2024



477



1



See all from Samar Singh

Recommended from Medium



In Level Up Coding by Bhargob Deka

Building a Hotel Recommender: Multi-Agent Framework with CrewAI, Ollama, and Gradio

A step-by-step guide on creating a multi-agent framework that can recommend hotels based on your specific criteria.



Jul 21, 2024



309



4



Rakesh Sheshadri

How CrewAI Was Built: Inside the Architecture and Codebase of CrewAI —Part 1

CrewAI Architecture Deep Dive :

5d ago 🖱 16

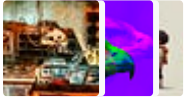


Lists



Generative AI Recommended Reading

52 stories · 1591 saves



What is ChatGPT?

9 stories · 491 saves



The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 534 saves



Natural Language Processing

1884 stories · 1529 saves



In Stackademic by Elinson

Lab #5: Chat with Multi-Agents (CrewAI, ChatGPT and Streamlit)

Chatting with multi-agents is as simple as you can



Aug 3, 2024



220



2





Lore Van Oudenhove

How to Build AI Agents with LangGraph: A Step-by-Step Guide

Learn how to build an AI assistant using LangGraph to calculate solar panel energy savings, showcasing advanced workflows, tools...

Sep 6, 2024



583



15




In Artificial Intelligence in Plain English by Okan Yenigün

Mastering CrewAI: Chapter 1—Your First Intelligent Workflow

The Basics of Agents and Tasks

★ Dec 10, 2024 🖱 123



 Samar Singh

How to Build AI Agents from Scratch with Python

Introduction to AI Agents

★ Aug 28, 2024 🖱 32



See more recommendations