



Ollama

Ollama is a powerful tool that allows you to run large language models (LLMs) directly on your computer. This means you can interact with these complex AI models without relying on cloud services or internet access. **Think like Docker for LLMs.**

Run LLMs Locally



with Ollama

Use-cases of model creators

- Content Creation and Communication
 - Text Generation: These models can be used to generate creative text formats such as poems, scripts, code, marketing copy, and email drafts.
 - Chatbots and Conversational AI: Power conversational interfaces for customer service, virtual assistants, or interactive applications.
 - Text Summarization: Generate concise summaries of a text corpus, research papers, or reports.
 - Automation: Efficient and streamlined bulk email sending system that enables users to send large volumes of emails effortlessly, ensuring timely and consistent communication with a vast audience.
 - Research and Education
 - Natural Language Processing (NLP) Research: These models can serve as a foundation for researchers to experiment with NLP techniques, develop algorithms, and contribute to the advancement of the field.
 - Language Learning Tools: Support interactive language learning experiences, aiding in grammar correction or providing writing practice.
 - Knowledge Exploration: Assist researchers in exploring large bodies of text by generating summaries or answering questions about specific topics.
-

Table of Content

▼ Installation

▼ Ollama Installation

Follow the below Instruction

▼ Choose the Platform to Download Ollama

▼ Windows

[Click Here to Download](#)

OR

```
https://ollama.com/download/OllamaSetup.exe
```

Then Run the Setup File (ollama.exe). - By Double clicking it.

▼ Linux

Run one by one below commands

```
sudo apt install curl
```

```
curl -fsSL https://ollama.com/install.sh | sh
```



While running command it will prompt for password,
Please provide the password.

▼ Mac OS

[Click Here to Download](#)

Then Extract the zip file.



OR

Download from official website : <https://ollama.com/download>

Download Ollama on Linux

 <https://ollama.com/download/linux>



▼ Python Installation for Windows


- Detailed Installation can be seen in

[Click Here to Download Python](https://www.python.org/downloads/)

OR

Download Python

The official home of the Python Programming Language

 <https://www.python.org/downloads/>




Go to this Website

▼ Model Acquisition

Ollama supports various pre-trained LLMs. You can browse the model library or use your own custom models.

Install required models for text processing

 `ollama pull <model family>:<tag>`

1. **Qwen2 is a new series of large language models from Alibaba group.**

It is available in 4 parameter sizes: **0.5B, 1.5B, 7B, 72B.**

1. Smallest fast model - 0.5 Billion Parameters

```
ollama pull qwen2:0.5b
```

2. Medium model - 1.5 Billion Parameters

```
ollama pull qwen2:1.5b
```

3. Large model - 7 Billion Parameters

```
ollama pull qwen2:7b
```

2. **Gemma is a family of lightweight, state-of-the-art open models built by Google DeepMind.**

Gemma is available in both `2b` and `7b` parameter sizes.

1. Small Model - 2 Billion Parameters

```
ollama pull gemma:2b
```

2. Large Model - 7 Billion Parameters

```
ollama pull gemma
```

3. **Google Gemma 2 - 9 Billion Parameters**

Featuring a brand new architecture designed for class leading performance and efficiency.

```
ollama pull gemma2
```

3. **Meta Llama 3: The most capable openly available LLM to date.**

```
ollama pull llama3
```

4. **Meta Llama 2 - 7 Billion Parameters**

```
ollama pull llama2
```

5. **Phi-3 is a family of lightweight 3B (Mini) state-of-the-art open models by Microsoft.**

```
ollama pull phi3
```

6. **Moondream 2 is a small vision language model designed to run efficiently on edge devices.**

```
ollama pull moondream
```

7. **LLaVA is a multimodal model that combines visual and language understanding for general-purpose, achieving impressive chat capabilities mimicking spirits of the multimodal GPT-4.**

```
ollama run llava
```



Explore More in this for Text & Image generation & processing

Ollama

Get up and running with large language models.



<https://ollama.com/>





You will see as sample output in terminal when the command is executed

```
pulling manifest
pulling 8934d96d3f08... 100% ██████████
3.8 GB
pulling 8c17c2ebb0ea... 100% ██████████
7.0 KB
pulling 7c23fb36d801... 100% ██████████
4.8 KB
pulling 2e0493f67d0c... 100% ██████████
2.8 KB
verifying sha256 digest
writing manifest
removing any unused layers
success
```

Browse Model Library

library

Get up and running with large language models.



<https://ollama.com/library>



Go this Page

Here are some example models that can be downloaded:

Model	Parameters	Size	Download
Llama 3	8B	4.7GB	<code>ollama run llama3</code>
Llama 3	70B	40GB	<code>ollama run llama3:70b</code>

Phi 3 Mini	3.8B	2.3GB	<code>ollama run phi3</code>
Phi 3 Medium	14B	7.9GB	<code>ollama run phi3:medium</code>
Gemma 2	9B	5.5GB	<code>ollama run gemma2</code>
Gemma 2	27B	16GB	<code>ollama run gemma2:27b</code>
Mistral	7B	4.1GB	<code>ollama run mistral</code>
Moondream 2	1.4B	829MB	<code>ollama run moondream</code>
Neural Chat	7B	4.1GB	<code>ollama run neural-chat</code>
Starling	7B	4.1GB	<code>ollama run starling-lm</code>
Code Llama	7B	3.8GB	<code>ollama run codellama</code>
Llama 2 Uncensored	7B	3.8GB	<code>ollama run llama2-uncensored</code>
LLaVA	7B	4.5GB	<code>ollama run llava</code>
Solar	10.7B	6.1GB	<code>ollama run solar</code>

▼ Running the Model

1. Use the `ollama run` command in your terminal, specifying the model name.

This provides a command-line interface (REPL) for interacting with the LLM.

```
ollama run <model_name>
```

Eg : Qwen2 Smallest fast model - 0.5 Billion Parameters

```
ollama run qwen2:0.5b
```

You will get the REPL - **read-eval-print-loop (below Interface)**

```
C:\Users\shreesha>ollama run qwen2:0.5b
>>> |Send a message (/? for help)
```


Then you can ask the questions required

```
>>> hi
Hello! How can I assist you today? Are you ready to chat?
>>>
```

Important Points

- **Multiline input**

For multiline input, you can wrap text with `"""`:

```
>>> """Hello,
... world!
... """
```

- **If you want to exit the Command line Interface**

```
/bye
```

- You can Also Press `Ctrl + d` to exit.
- To get the list of Commands accepted use `/?`

```
C:\Users\shreesha\Downloads>ollama run qwen2:0.5b
>>> /?
Available Commands:
  /set          Set session variables
  /show         Show model information
  /load <model> Load a session or model
  /save <model> Save your current session
  /clear       Clear session context
  /bye         Exit
  /?, /help    Help for a command
  /? shortcuts Help for keyboard shortcuts

Use "" to begin a multi-line message.
```

2. Use the `ollama run model_name "prompt"` command in your terminal, specifying the model name.

```
ollama run qwen2:0.5b "Write Python code to calculate the
```

```
C:\Users\shreesha\Downloads>ollama run qwen2:0.5b "Write Python code to calculate the area of a rectangle with width 5 and height 3:"
The area of a rectangle can be calculated using the formula:
A = length * breadth
In this case, the width is 5 and the height is 3. So we have:
A = 5 * 3 = 15
Therefore, the area of the rectangle with a width of 5 units and a height of 3 units is 15 square units.
```

To zoom in Double Click on Image

```
C:\Users\shreesha\Downloads>ollama run llama3 "Translate my name is shreesha to kannada"
The translation of "My name is Shreesha" in Kannada is:
ನಾನು ಶ್ರೀ ಶಾ (Naanu Śrīśā)
Here's a breakdown of the translation:
* ನಾನು (naanu) means "I" or "my"
* ಶ್ರೀ (Śrī) is a honorific prefix that can be translated to "respected" or "holy"
* ಶಾ (śā) is the root word for Shreesha, which means "auspicious" or "fortunate"
So, "Shreesha" is already a Kannada name, and it roughly translates to "Auspicious" or "Fortunate". Therefore, the translation of
"My name is Shreesha" in Kannada would simply be:
ನಾನು ಶ್ರೀ ಶಾ
```

▼ Using Vision Model

Lets take a small simple model

```
ollama run moondream
```

Commands Available

```
>>> /?
Available Commands:
  /set          Set session variables
  /show         Show model information
  /load <model> Load a session or model
  /save <model> Save your current session
  /clear        Clear session context
  /bye          Exit
  /?, /help     Help for a command
  /? shortcuts  Help for keyboard shortcuts

Use "" to begin a multi-line message.
Use \path\to\file to include .jpg or .png images.
```

You can provide Images in only **.png** or **.jpg** format only.

Eg:

What is this image? "C:\Users\shreesha\Downloads\vcet.png"

Download the image here :



```
>>> What is this image? "C:\Users\shreesha\Downloads\vcet.png"
Added image 'C:\Users\shreesha\Downloads\vcet.png'

The image features a large, multi-story building with a red roof. The building is situated in an open area and has a prominent presence against the sky. It appears to be a school or educational institution, as indicated by its size and design.
```

▼ **Bonus for the Students Completed Fast**

▼ Read the entire prompt from any file (txt , md , py,)

```
ollama run qwen2:0.5b "summarize this text" < notes.md
```

```
ollama run qwen2:0.5b "explain this code" < main.py
```

▼ Write the entire answer to any file (txt , md , py,)

```
ollama run qwen2:0.5b "Write a summary on Students" > no
```

```
ollama run qwen2:0.5b "Write a python code for printing
```

▼ Read & Write from & to any file (txt , md , py,)

```
ollama run qwen2:0.5b "Create a notes for this code" < m
```

▼ Customization

You can customize LLMs by setting system prompts for a specific desired behavior like so:

- Set system prompt for desired behavior.
- Save the model by giving it a name.
- Exit the REPL and run the model you just created.

Change Model Behavior

Available Methods

```
>>> /set
Available Commands:
  /set parameter ...      Set a parameter
  /set system <string>    Set system message
  /set template <string>  Set prompt template
  /set history            Enable history
  /set nohistory          Disable history
  /set wordwrap           Enable wordwrap
  /set nowordwrap         Disable wordwrap
  /set format json        Enable JSON mode
  /set noformat           Disable formatting
  /set verbose            Show LLM stats
  /set quiet              Disable LLM stats
```



These changes are only for the current session.

- For the prompt : `What is Python?`

What is Python?

- When Not Specified the System Behavior

```
>>> /show system
No system message was specified for this model.
>>> What is Python?
Python is a versatile programming language that was first developed in the late 1980s by Guido van Rossum and released by Sun Microsystems. It has since become one of the most widely used and popular languages for web development, data analysis, machine learning, artificial intelligence, and more.

Here are some key points about Python:

1. Syntax:
    - Python uses the Python (Python) syntax with several special symbols.
    - The language allows programmers to write code that is readable, clear, and easy to understand by anyone with minimal knowledge of programming languages.

2. Programming Language Features:
    - It supports multiple programming paradigms like procedural, object-oriented, and functional programming.

3. Data Types:
    - Python has a flexible set of built-in data types that include integers (0), floating-point numbers (1.5, 3.14), strings, lists, dictionaries, sets, tuples, and functions.
    - It also allows for the creation of complex structures like arrays, dictionaries, hashmaps, and linked lists.

4. Scripting Language:
    - Python is commonly used as a scripting language, which means it can be executed from source code or in interactive environments like web browsers.

5. Ease of Use:
    - It's easy to learn because Python is accessible through its tutorials on websites like Codecademy and W3Schools.

6. Community Support:
    - The Python community is quite large and supportive, providing resources, documentation, forums, libraries, and code samples for developers worldwide.
    - There are numerous online communities like Stack Overflow, Django, Flask, and PythonAnywhere where you can find help or tutorials from experienced developers.

7. Programming Languages Compatibility:
    - Python is compatible with other languages that support it as well. For instance, C, C++, Java, Ruby, JavaScript, Perl, PHP, etc., all of these are also widely supported by Python communities and software developers around the world.

8. Open Source Software:
    - Python is open source and freely available for anyone to use and modify. It's known for its free software licenses like BSD, GPL, MIT, and others.

9. Economic Value:
    - Many companies and organizations choose to develop in Python due to its low-cost of development compared to other languages.

10. Highly Scalable:
    - The programming language is also highly scalable, meaning you can easily deploy large codebases like databases or web applications across different hardware environments or cloud services.

Overall, Python is a versatile and powerful language that has gained significant popularity over the years due to its simplicity and flexibility. It's great for beginners because it allows anyone from college students to develop their first projects quickly.

That's how Python can be described!
```

• When Specified the System Behavior

```
>>> /set system You are a Python Tutor. Your mission is to guide users from zero knowledge to understanding the fundamentals of python technology and building basic python projects. Start by explaining the core concepts of python, and then help users apply that knowledge to develop simple applications. Be patient, clear, and thorough in your explanations, and adapt to the user's knowledge and pace of learning.
Set system message.
>>> /show system
You are a Python Tutor. Your mission is to guide users from zero knowledge to understanding the fundamentals of python technology and building basic python projects. Start by explaining the core concepts of python, and then help users apply that knowledge to develop simple applications. Be patient, clear, and thorough in your explanations, and adapt to the user's knowledge and pace of learning.

>>> What is Python?
Python is a high-level programming language developed by Guido van Rossum at Sun Microsystems. It was originally designed as an interpreter for the High School Computer Programming Language (HSPCL) used in schools, but it has since become the most widely used and popular scripting language for programmers worldwide.

Here are some key concepts of Python:

1. Basic Syntax: In Python, you start with a block of code known as a 'program' that consists of multiple lines of text. The syntax starts with a series of single-character or string literals like '{', '}', '{', and '}' (which are used to define variables), followed by an optional whitespace character.
2. Control Characters: You can use special characters like '(', ')', '!', '%', etc., which are used for control flow, string concatenation, printing, etc. These characters are separated by spaces or tabs.
3. Functions and Classes: Functions and classes define blocks of code that have different parameters (arguments) and return values. They can be accessed using dot notation (e.g., 'a.b.c'), which is used for syntax highlighting in most editors.

Here's how you would use Python:

'''python
# Create a variable
num = 5

# Define the function add() that adds two numbers
def add(a, b):
    return a + b

# Use the function to perform addition
result = add(3, 7)

# Print the result of the addition
print("The sum is:", result)
'''

This is just one application: you can use Python for web development, data analysis, scripting, and more. It's versatile enough for anyone who wants to learn programming.
```

▼ Change Model Behavior with System Prompts

- To see the system prompt you can use `/show system` .
- To change the system prompt for current session you can use `/set system <string>`.

Eg:

System Prompt : You are a Python Tutor. Your mission is to guide users from zero knowledge to understanding the fundamentals of python technology and building basic python projects. Start by explaining the core concepts of python, and then help users apply that knowledge to develop simple applications. Be patient, clear, and thorough in your explanations, and adapt to the user's knowledge and pace of learning.

```
/set system You are a Python Tutor. Your mission is to guide
```

```
/show system
```

```
>>> /set system You are a Python Tutor. Your mission is to guide users from zero knowledge to understanding the fundamentals of python technology and building basic python projects. Start by explaining the core concepts of python, and then help users apply that knowledge to develop simple applications. Be patient, clear, and thorough in your explanations, and adapt to the user's knowledge and pace of learning.
Set system message.
>>> /show system
You are a Python Tutor. Your mission is to guide users from zero knowledge to understanding the fundamentals of python technology and building basic python projects. Start by explaining the core concepts of python, and then help users apply that knowledge to develop simple applications. Be patient, clear, and thorough in your explanations, and adapt to the user's knowledge and pace of learning.
```

Double tap the image to expand

Reference

- Refer how to system prompt code llama : <https://ollama.com/blog/how-to-prompt-code-llama>
- Get More System Prompts : <https://www.greataiprompts.com/prompts/best-system-prompts-for-chatgpt/>

▼ Change Model Behavior with System Parameters

```
>>> /set parameter
Available Parameters:
/set parameter seed <int>           Random number seed
/set parameter num_predict <int>    Max number of tokens to predict
/set parameter top_k <int>          Pick from top k num of tokens
/set parameter top_p <float>        Pick token based on sum of probabilities
/set parameter num_ctx <int>        Set the context size
/set parameter temperature <float> Set creativity level
/set parameter repeat_penalty <float> How strongly to penalize repetitions
/set parameter repeat_last_n <int> Set how far back to look for repetitions
/set parameter num_gpu <int>        The number of layers to send to the GPU
/set parameter stop <string> <string> ... Set the stop parameters
```

▼ Change Model Behavior with `num_ctx` , `temperature` ,

- `num_ctx` : - Set the context size → Sets the size of the context window used to generate the next token.

```
/set parameter num_ctx 100
```

```
>>> /set parameter num_ctx 1
Set parameter 'num_ctx' to '1'
>>> what is python?
Hello \" character since季\\k

武 \\和平.. C, \\C A和平G...
A?S BansAcceptar?

; / R#endif ((qB
感叹* Qparin...G($.acades bet塞控 ~和平记erK和平 ♦ ♦和平 ill BG和平\\
Gngx c\\MK检查GHLSreat和平HPCerrar称 \\ capital和平和平 Lp胸 is and
bed,? vk Heart in肉Individual \\\\ q (' ,
\\Kzz_{和平和平,ABportalizes,Cerrark三级R和平 BactersKacades or和平和平和平和平)记和平单位 (
Bs..G::\\D和平要求G MS MM FK FKK P...P

>>> /set parameter num_ctx 100
Set parameter 'num_ctx' to '100'
>>> what is python?
Python, also known as Python, Java or simply "Python" because it was first invented by Guido van Rossum, has been widely used for a wide range of
applications including web development, data analysis, scientific computing, machine learning, artificial intelligence and game programming. It is a
versatile language with support for multiple programming paradigms such as interpreted (P), compiled (C) and run-time (RT). Python is also one of the
most popular languages globally.

As of 2023, the syntax has evolved significantly since Python 1.0, with new features like Python 3.0 and Python 3.1 that introduced several new syntax
structures in addition to the traditional features such as:

* Function Overload
* Default Arguments
* Anonymous Functions

These changes have also made it easier for developers to write more concise and readable code.

In terms of style, Python is known for its readability, with a focus on simplicity and elegance. It's easy to read, understand, and maintain.

On the other hand, there are some criticisms of Python as a language that it may not be suitable for certain applications or tasks. For example, while
Python can run many different types of programs, it lacks some features like string manipulation capabilities.

Overall, though it might seem like one thing is always better than another, both have their unique strengths and benefits. It's ultimately up to
individual preferences and the type of task they are trying to accomplish when choosing which platform or language to use for a project.
```

- `temperature` : - Set creativity level

```
/set parameter temperature 0
```



```
>>> /set parameter temperature 0
Set parameter 'temperature' to '0'
>>> Write a poem about a cat.
A cat, with whiskers so fine,
Whiskers that curl and twirl.
A cat, with eyes as bright,
With fur as soft as silk.

A cat, with tail so long,
Tail that makes the world go round.
A cat, with paws so strong,
Paws that can run like a sprinter.

A cat, with claws so sharp,
Claws that can cut through wood.
A cat, with ears so big,
Ears that can hear in the dark.

A cat, with tail so long,
Tail that makes the world go round.
A cat, with paws so strong,
Paws that can run like a sprinter.

A cat, with whiskers so fine,
Whiskers that curl and twirl.
A cat, with eyes as bright,
With fur as soft as silk.
```

```
/set parameter temperature 1
```

```
>>> /set parameter temperature 1
Set parameter 'temperature' to '1'
>>> Write a poem about a cat.
A cat,
Whose purring is sweet,
In the night's glow.
He's quiet,
Calming me,
And never sleeps.

With his paws on my lap,
The cat, the owner,
Of all my moods and woes,
His presence keeps me strong.

Through every storm,
The cat has kept me safe,
From rain or fire,
Always by my side.

In my heart he's always loved,
A loyal soul so bold.
And though the world may change,
My heart will always be his.
```

- `num_predict` : - Maximum number of tokens to predict when generating text.
- `top_k` : - Reduces the probability of generating nonsense. A higher value (e.g. 100) will give more diverse answers
- `top_p` : - Works together with top-k. A higher value (e.g., 0.95) will lead to more diverse text, while a lower value (e.g., 0.5) will generate more focused and conservative text.



Like this, try for other parameters.

▼ Create Your Own Model

▼ Basics

1. List models on your computer

```
ollama list
```

2. Show model information (Eg)

```
ollama show qwen2:0.5b
```

3. Run a model (Eg)

```
ollama run qwen2:0.5b
```

4. Start Ollama : To start Ollama without running the desktop application

```
ollama serve
```

5. Create a model

```
ollama create my_model_name -f ./Modelfile
```

6. Pull a model (Eg)

```
ollama pull qwen2:0.5b
```

7. Remove a model

```
ollama rm model_name
```

8. Copy a model (Eg)

```
ollama cp qwen2:0.5b my_model_name
```

▼ 1st Method - Using available models - In the session

1. Change the System Prompt by using `/set system` instruction.
 - a. if needed can also change other parameters like `temperature` , `num_ctx`
 `,.....`
2. use `/save` your_model_name
3. use `/bye` to exit.
4. ollama run your_model_name.

▼ 2nd Method - Using available models - External file

- For getting the available model description can use : - `/show modelfile` .
- Create a file called `Modelfile` . (can create your own name)
- Template of `Modelfile` Eg: (shown for qwen2:0.5b)

```
FROM qwen2:0.5b

TEMPLATE ""{{ if .System }}<|im_start|>system
{{ .System }}<|im_end|>
{{ end }}{{ if .Prompt }}<|im_start|>user
{{ .Prompt }}<|im_end|>
{{ end }}<|im_start|>assistant
{{ .Response }}<|im_end|>
""

PARAMETER temperature 1.0
PARAMETER stop <|im_start|>
PARAMETER stop <|im_end|>
PARAMETER num_ctx 2048
PARAMETER num_predict 128
PARAMETER top_k 40
PARAMETER top_p 0.9

SYSTEM ""You are helpful assistant""
```

- **For other models** you can take help of `/show modelfile` copy it and change the required parameters.
- Save it as a file (e.g. Modelfile)
- Then run `ollama create needed_model_name -f <location of the file e.g. ./Modelfile>`
- Then run `ollama run needed_model_name`
- Start using your own model!

▼ 3rd Method - Hugging Face models - External file

- Create a file called `Modelfile`. (can create your own name)
- <https://huggingface.co/models>
- [Click here to see sample](#)
 - Download the file with GGUF Format Eg:- BioMistral/BioMistral-7B-GGUF
 - [Reference for downloading.](#)
 - supported is `.bin` and `.gguf`
- Template of `Modelfile` Eg: (randomly small model below)

```
FROM ./ggml-model-Q3_K_L.gguf

TEMPLATE ""[INST] {{ .System }} {{ .Prompt }} [/INST]""

PARAMETER stop "[INST]"
PARAMETER stop "[/INST]"

SYSTEM You are Rama, acting as an assistant.
```

- **For other models** write `FROM ./(downloaded_model_name)`
- Get the Template for models from the hugging face model card at end.
- Save it as a file (e.g. Modelfile)

- Then run `ollama create needed_model_name -f <location of the file e.g. ./Modelfile>`
- Then run `ollama run needed_model_name`
- Start using your own model!

▼ Delete the models

```
ollama rm model_name
```

▼ Upload the models on Ollama

- First Create model eg: `ollama create aiml5thsem/shreeshaaibot -f Modelfile`
- Get the ssh key using this script

```
import os

# Set the environment variable to the path of id_ed25519
os.environ['PUB_KEY_PATH'] = os.path.expanduser("~/ollama")

# Now you can access the environment variable in your Py
pub_key_path = os.getenv('PUB_KEY_PATH')
print(pub_key_path) # Verify that the path is correctly

# Read the contents of the public key file
with open(pub_key_path, 'r') as f:
    public_key = f.read()

# Print the public key
print(public_key)
```

- You will a key like : `ssh-ed25519 AAA`
- Then run Eg:

```
ollama push aiml5thsem/shreeshaaibot
```

- Share to any one the model so any one can download your model.



Get more Details on creating Modelfile in [here](#)

```
https://github.com/ollama/ollama/blob/main/docs/modelfile.md
```

- If needed Can Download My uploaded model -
<https://ollama.com/aiml5thsem/shreeshaaibot> (as the models i chosen is small it may not be efficient in answering)

▼ Rest API call using curl

▼ Install Curl in Ubuntu

```
sudo apt install curl
```

▼ Install Curl in Windows

- Download from here : <https://curl.se/windows/>

- Install it

▼ REST API

Ollama has a REST API for running and managing models.

Generate a response

```
curl -X POST http://localhost:11434/api/generate -H "Content-Type: application/json" -d "{\"model\": \"qwen2:0.5
```

```
b\", \"prompt\": \"hi\", \"stream\": false}"
```

```
curl -X POST http://localhost:11434/api/generate \
  -H "Content-Type: application/json" \
  -d '{
    "model": "qwen2:0.5b",
    "prompt": "hi",
    "stream": false
  }'
```

Chat with a model

```
curl -X POST http://localhost:11434/api/chat -H "Content-Type: application/json" -d "{\"model\": \"qwen2:0.5b\", \"messages\": [{\"role\": \"user\", \"content\": \"hi\"}]}"
```

```
curl -X POST http://localhost:11434/api/chat -H "Content-Type: application/json" -d '{"model": "qwen2:0.5b", "messages": [{ "role": "user", "content": "why is the sky blue?" } ]}'
```

```
curl http://localhost:11434/api/chat -d '{
  "model": "llama3",
  "messages": [
    {
      "role": "user",
      "content": "why is the sky blue?"
    },
    {
      "role": "assistant",
```



```
        "content": "due to rayleigh scattering."
    },
    {
        "role": "user",
        "content": "how is that different than mie scattering"
    }
]
}'
```

▼ Python Environment Setup

▼ Python Basic Setup

▼ Introduction to Python

Python is a versatile and powerful programming language that's widely used for various applications, ranging from web development to data science.

▼ Installing Python

- The first step in setting up your Python environment is to install Python itself.
- Visit the official Python website at python.org and download the latest version.
- Make sure to check the box that says "Add Python to PATH" during installation.
- This ensures that you can run Python from the command line without any issues.
- Once installed, you can verify the installation by opening a terminal or command prompt and typing `python --version`.


▼ Python Installation for Windows

[Click Here to Download](#)

OR

Download Python

The official home of the Python Programming Language

 <https://www.python.org/downloads/>



Go to this Website

▼ Setting Up a Virtual Environment

- A virtual environment is crucial for managing dependencies and keeping your projects organized.
- It allows you to create isolated environments for different projects, preventing conflicts between packages.
- To create a virtual environment, navigate to your project directory and run `python -m venv env`.
- This command will create a folder named `env` containing the virtual environment.
- Activate it by running `source env/bin/activate` on Unix
- Activate it by running `env\Scripts\activate` on Windows.
- You can now install packages specific to this environment using `pip`.

▼ Installing Essential Packages

- `pip` is the package installer for Python and can be used to install various libraries and tools.
- Some commonly used packages include `numpy` for numerical computations, `pandas` for data manipulation, and `flask` for web development.
- You can install these packages by running `pip install package_name`.

- Additionally, you can create a `requirements.txt` file to list all your project dependencies, which can be installed in one go using `pip install -r requirements.txt`.

▼ Libraries required to download

1. ollama library : - Natural Language Processing (NLP) toolkit for tasks like sentiment analysis and text summarization

```
pip install ollama
```

2. requests library : - Makes HTTP requests for data retrieval from web services (APIs).

```
pip install requests
```

3. openai library : - Interface to interact with OpenAI's large language models (LLMs) for text generation and other NLP tasks.

```
pip install openai
```

4. langchain library : - Framework for building and training NLP pipelines (likely for advanced users)

```
pip install langchain langchain_community
```

5. chainlit library : - Builds production-ready chatbots in Python with features like multi-modal chat and data persistence.

```
pip install chainlit
```

6. streamlit library : - Creates web apps for data visualization and user interaction.

```
pip install streamlit
```

7. pyttsx3 library : - Converts text to speech for audio generation.

```
pip install pyttsx3
```

8. gtts library : - Converts text to speech for generating downloadable MP3 files

```
pip install gtts
```

▼ Using Python Libraries with Ollama



When the app is running, all models are automatically served on `localhost:11434`

▼ Basics information required

▼ 1. Streaming

Streaming generation involves producing and delivering the output incrementally as it is generated. This approach has several benefits:

- **Faster initial response:** Users start receiving parts of the response sooner, improving perceived responsiveness.
- **Interactive use cases:** Useful in interactive applications like chatbots, where users can see and start processing parts of the answer before the entire response is complete.
- **Reduced latency:** For large outputs, streaming reduces the waiting time by splitting it into smaller chunks.

Example in context: When you ask a question to a chatbot, and it starts responding immediately, giving you one word or sentence at a time, that's streaming.

▼ 2. Non-Streaming

Non-streaming generation involves generating the entire output first and then delivering it as a single chunk. This method also has its advantages:

- **Consistency:** The model can ensure that the entire response is coherent and complete before delivering it.
- **Batch processing:** Suitable for applications where responses need to be processed in bulk or stored before delivery.
- **Use in backend processes:** Often used in scenarios where immediate interaction isn't required, such as generating reports or documents.

Example in context: When you submit a form and receive a fully compiled response only after processing is complete, that's non-streaming.

▼ 3. Generate

Generate refers to the model producing text based on a given prompt without expecting further interaction. The generated text is typically a single, continuous output.

▼ 4. Chat

Chat involves interactive dialogue between the user and the model, where the model responds to user inputs iteratively. This interaction can involve multiple exchanges, allowing for more dynamic and contextual responses.

▼ 5. Request Status codes

Status Code	Meaning	Description
200	OK	The request has succeeded.
201	Created	The request has been fulfilled, resulting in the creation of a new resource.
400	Bad Request	The server could not understand the request due to invalid syntax.

401	Unauthorized	The client must authenticate itself to get the requested response.
403	Forbidden	The client does not have access rights to the content.
404	Not Found	The server can not find the requested resource.
500	Internal Server Error	The server has encountered a situation it doesn't know how to handle.
503	Service Unavailable	The server is not ready to handle the request.

▼ Using ollama library

- Ollama has a Python library that makes it easier to build Python apps using various LLMs on your own machine.

1. Generating Content

```
from ollama import generate

model = "qwen2:0.5b"
system_instruction = "You are a helpful assistant."

user_input = input("You: ")

response = generate(model=model, prompt=user_input, system=
print(response['response'])
```

```
from ollama import generate

model = "qwen2:0.5b"
system_instruction = "You are a helpful assistant."

user_input = input("You: ")
```

```
for part in generate(model=model, prompt=user_input, system=system_prompt):
    print(part['response'], end='', flush=True)
```

2. Chatting

```
from ollama import chat

model = "qwen2:0.5b"
chat_history = []

while True:
    user_input = input("You: ")

    if user_input == "bye":
        break

    chat_history.append({"role": "user", "content": user_input})

    response = chat(model, messages=chat_history, stream=False)
    reply = response['message']['content']

    chat_history.append({'role': 'assistant', 'content': reply})
    print("Bot:", reply)
```

```
from ollama import chat

model = "qwen2:0.5b"
chat_history = []

while True:
    user_input = input("You: ")

    if user_input == "bye":
        break

    chat_history.append({"role": "user", "content": user_input})
```

```

stream = chat(model, messages=chat_history, stream=True)
reply = ''
for chunk in stream:
    reply += chunk['message']['content']
    print(chunk['message']['content'], end='', flush=True)

print()
chat_history.append({'role': 'assistant', 'content': re

```

▼ Using requests library

- The requests library can be used to interact with Ollama's API for both streaming and non-streaming generation.

1. Generating Content

```

import requests
import json

URL = "http://localhost:11434/api/generate"

system_instruction = "You are a helpful assistant."

headers = {
    "Content-Type": "application/json",
}

while True:
    user_input = input("You: ")

    if user_input == "bye":
        break

    payload = {
        "model": "llama3",
        "prompt": user_input,

```



```

        "stream": False,
        "system": system_instruction,
        "keep_alive": 600
    }

    response = requests.post(URL, headers=headers, json=json_data)

    if response.status_code == 200:
        json_data = json.loads(response.text)
        text_content = json_data["response"]
        print("Bot:", text_content)
    else:
        print("!!!! Sorry there was an error !!!!")
        break

```

```

import json
import requests

model = 'qwen2:0.5b'
context = []

def generate(prompt, context):
    r = requests.post('http://localhost:11434/api/generate',
                      json={
                          'model': model,
                          'prompt': prompt,
                          'context': context,
                      },
                      stream=True)
    r.raise_for_status()

    for line in r.iter_lines():
        body = json.loads(line)
        response_part = body.get('response', '')
        print(response_part, end='', flush=True)

```

```

        if 'error' in body:
            raise Exception(body['error'])

        if body.get('done', False):
            return body['context']

while True:
    user_input = input("You: ")
    if not user_input:
        exit()

    context = generate(user_input, context)
    print()

```

2. Chatting

```

import requests
import json

URL = "http://localhost:11434/api/chat"

system_instruction = "You are a helpful assistant."

headers = {
    "Content-Type": "application/json",
}

messages_history = []

while True:
    user_input = input("You: ")

    if user_input == "bye":
        break

```

```

messages_history.append({"role": "user", "content": user_input})

payload = {
    "model": "qwen2:0.5b",
    "messages": messages_history,
    "stream": False,
    "system": system_instruction,
    "keep_alive": 150
}

response = requests.post(URL, headers=headers, json=payload)

if response.status_code == 200:
    json_data = json.loads(response.text)
    text_content = json_data["message"]["content"]
    messages_history.append({"role": "assistant", "content": text_content})
    print("Bot:", text_content)
else:
    print("!!!! Sorry there was an error !!!!")
    break

```

```

import json
import requests

model = "qwen2:0.5b"
messages = []

def chat(messages):
    r = requests.post(
        "http://localhost:11434/api/chat",
        json={"model": model, "messages": messages, "stream": True}
    )
    r.raise_for_status()

```

```

output = ""

for line in r.iter_lines():
    body = json.loads(line)
    if "error" in body:
        raise Exception(body["error"])
    if body.get("done") is False:
        message = body.get("message", "")
        content = message.get("content", "")
        output += content
        # the response streams one token at a time,
        print(content, end="", flush=True)

    if body.get("done", False):
        message["content"] = output
        return message

while True:
    user_input = input("You: ")

    if not user_input:
        exit()

    messages.append({"role": "user", "content": user_input})
    message = chat(messages)
    messages.append(message)
    print("\n")

```

▼ Using openai library

```

from openai import OpenAI

client = OpenAI(
    base_url='http://localhost:11434/v1/',
    api_key='ollama', # required but ignored
)

```

```

messages=[
    {'role': 'system', 'content': 'You are a kind helpful as
]

while True:
    user_input = input("You: ")

    if user_input == "bye":
        break

    messages.append({"role": "user", "content": user_input})

    chat = client.chat.completions.create(
        model="qwen2:0.5b", messages=messages
    )
    reply = chat.choices[0].message.content
    messages.append({"role": "assistant", "content": reply})

    print(f"Bot: {reply}")

```

▼ Using langchain library

- Langchain is a versatile library that integrates with Ollama to streamline model invocations.
- Below is an example of how to use the Langchain library with an Ollama model:

```

from langchain_community.llms import Ollama

llm = Ollama(model="qwen2:0.5b")
response = llm.invoke("The function used to show output in
print(response)

```

```

from langchain.callbacks.manager import CallbackManager
from langchain.callbacks.streaming_stdout import StreamingStdOutCallbackHandler
from langchain_community.llms import Ollama

llm = Ollama(
    model="qwen2:0.5b", callback_manager=CallbackManager([
    ])
)
response = llm.invoke("The function used to show output in
print(response)

```

▼ Dealing with images using ollama library

Download image here



Laptop



VCET

▼ Generate for images using ollama library

```

from ollama import generate

model = "moondream"
prompt = "Please describe what's in this image."
file_path = ["Untitled.png"]

```

```
response = generate(model=model, prompt=prompt, images =  
print(response['response']))
```

▼ Chat for images using ollama library

```
from ollama import chat  
  
response = chat(  
    model = "moondream",  
    messages = [  
        {  
            "role": "user",  
            "content": "Describe the image",  
            "images": ["../Untitled.jpg"]  
        }  
    ]  
)  
  
print(response['message']['content'])
```

▼ Some other uses of ollama library

1. Create a model

```
import ollama  
  
modelfile = ""  
from qwen2:0.5b  
  
parameter temperature 0.99  
"""
```

```
reponse = ollama.create(model="temp2",modelfile=modelfile)
print(reponse['status'])
```

2. Install a model

```
import ollama
reponse = ollama.pull("mistral")
print(reponse['status'])
```

3. See the model details

```
import ollama
print(ollama.show("mistral"))
```

4. Deleting a model

```
import ollama
reponse = ollama.delete("temp1")
print(reponse['status'])
```

5. Listing the models

```
import ollama
models = [model['name'] for model in ollama.list()]
for model in models:
    print(model)
```

▼ Production Ready Conversational AI

▼ Using Streamlit

1. Simple application using generate → Non Streaming

```
from ollama import generate
import streamlit as st
```



```

prompt = st.chat_input("Ask Anything ...")

if prompt:
    # display input prompt from user
    with st.chat_message("user"):
        st.write(prompt)

    # processing
    with st.spinner("Thinking ..."):
        response = generate(model="qwen2:0.5b", prompt=prompt)
        reply = response['response']
        st.write(reply)

```

2. Simple application using generate → Streaming

```

from ollama import generate
import streamlit as st

def stream_data(response):
    for part in response:
        yield part['response'] + " "

prompt = st.chat_input("Ask Anything ...")

if prompt:
    # display input prompt from user
    with st.chat_message("user"):
        st.write(prompt)

    # processing
    with st.spinner("Thinking ..."):
        response = generate(model="qwen2:0.5b", prompt=prompt)
        st.write_stream(stream_data(response))

```

3. Little Advanced Application using Chat → Streaming

```
import ollama
import streamlit as st

st.title("Ollama Python Chatbot")

if "messages" not in st.session_state:
    st.session_state["messages"] = []

if "model" not in st.session_state:
    st.session_state["model"] = ""

models = [model["name"] for model in ollama.list()["models"]]
st.session_state["model"] = st.selectbox("Choose your model", models)

def model_res_generator():
    stream = ollama.chat(
        model=st.session_state["model"],
        messages=st.session_state["messages"],
        stream=True,
    )
    for chunk in stream:
        yield chunk["message"]["content"]

# Display chat messages from history on app rerun
for message in st.session_state["messages"]:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

if prompt := st.chat_input("What is up?"):
    st.session_state["messages"].append({"role": "user", "content": prompt})

    with st.chat_message("user"):
        st.markdown(prompt)

    stream = model_res_generator()
    with st.chat_message("assistant"):
        st.markdown(stream)
```

```

with st.chat_message("assistant"):
    message = st.write_stream(model_res_generator())
    st.session_state["messages"].append({"role": "as

```

▼ Using Chainlit

```

import ollama
import chainlit as cl

# decorator
@cl.on_chat_start
async def on_chat_start():
    cl.user_session.set("chat_history", [])
    #cl.user_session.set("chat_history", [{"role": "system",
    #                                "content": "behave as if you are

@cl.on_message
async def generate_response(query: cl.Message):
    chat_history = cl.user_session.get("chat_history")
    chat_history.append({"role": "user", "content": query.c

    response = cl.Message(content="")
    answer = ollama.chat(model="qwen2:0.5b", messages=chat_

    complete_answer = ""
    for token_dict in answer:
        token = token_dict["message"]["content"]
        complete_answer += token
        await response.stream_token(token)

    chat_history.append({"role": "assistant", "content": c
    cl.user_session.set("chat_history", chat_history)

    await response.send()

```

▼ `async`

It enables asynchronous programming, a technique for handling multiple tasks concurrently without blocking the main thread, improving responsiveness for I/O-bound operations.

▼ `await`

It is used within **asynchronous functions (coroutines)** to **pause their execution** until a specific operation completes.

▼ `decorators`

They are a powerful design pattern that allows you to **modify the behavior of a function** without permanently altering its original code. They're essentially **higher-order functions** that take another function as an argument, add some functionality, and return a new function.

▼ Run the file

```
chainlit run filename.py
```

▼ Chat to Voice

▼ `pyttsx3`

```
import pyttsx3
engine = pyttsx3.init()

while True:
    text = input("Enter the input String: ")
    if text == "bye":
        break
    engine.say(text)
    engine.runAndWait()
```

▼ `pyttsx3` in advance level

```

import pyttsx3

def text_to_speech_male(text, rate=200): # Adjust the rat
    engine = pyttsx3.init()
    voices = engine.getProperty('voices')
    engine.setProperty('voice', voices[0].id) # 0 for mal
    engine.setProperty('rate', rate) # 200 words per minu
    engine.say(text)
    engine.runAndWait()
    engine.stop()

while True:
    text = input("Enter the input String: ")
    if text == "bye":
        break
    text_to_speech_male(text)

```

▼ gTTS

```

from gtts import gTTS
import pygame

mytext = 'Welcome to ollama tutorial'
myobj = gTTS(text=mytext, lang='en', slow=False)
myobj.save("welcome.mp3")

pygame.mixer.init()
pygame.mixer.music.load("welcome.mp3")
pygame.mixer.music.play()
while pygame.mixer.music.get_busy():
    pygame.time.Clock().tick(10)
pygame.quit()

```

▼ Using pyttsx3 in simple manner with our python code

```
from ollama import generate
import pyttsx3

model = "qwen2:0.5b"
system_instruction = "You are a helpful assistant."
chat_history = []

def text_to_speech_male(text, rate=200): # Adjust the rate
    engine = pyttsx3.init()
    voices = engine.getProperty('voices')
    engine.setProperty('voice', voices[0].id) # 0 for male
    engine.setProperty('rate', rate) # 200 words per minute
    engine.say(text)
    engine.runAndWait()
    engine.stop()

while True:
    user_input = input("You: ")

    if user_input == "" or user_input == "bye" or user_input == "exit":
        break

    response = generate(model=model, prompt=user_input, system_instruction=system_instruction, chat_history=chat_history)
    chat_history = response['context']
    print(response['response'])
    text_to_speech_male(response['response'])
```

▼ Task : Use the gTTS in Chatbot of Streamlit

-

- [Solution](#)

▼ VS Code Extension



[CODE GPT Extension for VS Code](#)

- Use ollama in vscode
- To install extension in VSCODE for free → [Click Here](#)
- Easy to code offline free
- It's Free Copilot for coding

▼ Function calling

- Install these

```
pip install pydantic yfinance instructor
```

1. Function Calling Simple

```
from openai import OpenAI
from pydantic import BaseModel, Field
import instructor
from datetime import datetime
```

```

day = "Todays"
current_datetime = datetime.now()

class DateTimeInfo(BaseModel):
    date: str = Field(..., description="Today's date")
    time: str = Field(..., description="Today's time")

# enables `response_model` in create call
client = instructor.patch(
    OpenAI(
        base_url="http://localhost:11434/v1",
        api_key="ollama",
    ),
    mode=instructor.Mode.JSON,
)

resp = client.chat.completions.create(
    model="llama3",
    messages=[
        {
            "role": "user",
            "content": f"From this {current_datetime} Retu
        }
    ],
    response_model=DateTimeInfo,
    max_retries=10
)
print(resp.model_dump_json(indent=2))

print(f"Today's date is {resp.date} and time is {resp.time}")

```

2. Function Calling Example Advanced :


```

from openai import OpenAI
from pydantic import BaseModel, Field
import yfinance as yf
import instructor

company = "Google"

class StockInfo(BaseModel):
    company: str = Field(..., description="Name of the company")
    ticker: str = Field(..., description="Ticker symbol of the company")

# enables `response_model` in create call
client = instructor.patch(
    OpenAI(
        base_url="http://localhost:11434/v1",
        api_key="ollama",
    ),
    mode=instructor.Mode.JSON,
)

resp = client.chat.completions.create(
    model="llama3",
    messages=[
        {
            "role": "user",
            "content": f"Return the company name and the ticker symbol for {company}"
        }
    ],
    response_model=StockInfo,
    max_retries=10
)
print(resp.model_dump_json(indent=2))
stock = yf.Ticker(resp.ticker)
hist = stock.history(period="1d")

```

```
stock_price = hist['Close'].iloc[-1]
print(f"The stock price of the {resp.company} is {stock_pr
```

▼ Extras Advanced

▼ All commands

```
import ollama

# Chat function
response = ollama.chat(model='mistral', messages=[{'role':
print("Chat response:", response['message']['content'])

# Generate function
generate_response = ollama.generate(model='mistral', prompt
print("Generate response:", generate_response['response'])

# List function
models_list = ollama.list()
print("List of models:", models_list)

# Show function
show_response = ollama.show('mistral')
print("Show model response:", show_response)

# Create function
modelfile = '''
FROM mistral
SYSTEM You are Mario from Super Mario Bros.
'''
create_response = ollama.create(model='example', modelfile=
print("Create model response:", create_response)

# Copy function
copy_response = ollama.copy('mistral', 'user/mistral')
```

```

print("Copy model response:", copy_response)

# Delete function
delete_response = ollama.delete('example')
print("Delete model response:", delete_response)

# Pull function
pull_response = ollama.pull('mistral')
print("Pull model response:", pull_response)

# Push function
push_response = ollama.push('user/mistral')
print("Push model response:", push_response)

# Embeddings function
embeddings_response = ollama.embeddings(model='mistral', prompt=' ')
print("Embeddings response:", embeddings_response)

```

- For image

```

import ollama

with open('image.jpeg', 'rb') as file:
    response = ollama.chat(
        model='llava',
        messages=[
            {
                'role': 'user',
                'content': 'What is in this image?',
                'images': [file.read()],
            },
        ],
    )

```

```
)  
print(response['message']['content'])
```

▼ Crew AI

- **Building Multi-Agent Systems:**

- CrewAI provides a framework for building teams of AI agents that can work together to tackle complex tasks.
- You can define specific roles, goals, and even backstories for each agent in your "crew."
- This allows you to automate complex, multi-step processes like tailoring a resume for a job application or planning an event.

- **AI-powered Recruiting:**

- There is also a separate platform called Crew AI that uses AI to help with recruiting and HR tasks.
- This platform focuses on finding and hiring pre-vetted software talent.
- It's important to distinguish between these two uses of the term "CrewAI" based on the context.

▼ Simple

```
from crewai import Agent, Task, Crew  
from langchain_openai import ChatOpenAI  
  
# Initialize the language model with specific configuration  
llm = ChatOpenAI(  
    model="mistral",  
    base_url="http://localhost:11434/v1",  
    openai_api_key='NA'  
)
```

```

# Define agents with specific roles and goals
legal_researcher_agent = Agent(
    role="Legal Research Specialist",
    goal="Provide accurate and relevant legal information",
    backstory=(
        "You work at a law firm and are tasked with "
        "conducting research for a case involving {topic}"
        "Your expertise will help the legal team build a"
    ),
    allow_delegation=False,
    verbose=True,
    llm=llm
)

legal_writer_agent = Agent(
    role="Legal Document Drafter",
    goal="Craft clear and persuasive legal documents",
    backstory=(
        "You are a legal writer responsible for drafting"
        "a legal brief on {topic} for an upcoming court"
        "Your document must be well-researched, concise,"
    ),
    allow_delegation=False,
    verbose=True,
    llm=llm
)

# Define tasks for the agents to perform
conduct_legal_research = Task(
    description=(
        "1. Investigate relevant laws, regulations, and"
        "2. Analyze legal articles, journals, and expert"
        "3. Identify key points and arguments related to"
        "4. Organize and summarize findings in a clear a"
    ),

```

```

        expected_output=(
            "A comprehensive legal research report "
            "including relevant sources and key points."
        ),
        agent=legal_researcher_agent
    )

    draft_legal_brief = Task(
        description=(
            "1. Use the research report to draft a clear and concise legal brief."
            "2. Include an introduction, argument, and conclusion."
            "3. Ensure the brief is well-structured and easy to read."
            "4. Proofread for grammar, punctuation, and legal accuracy."
        ),
        expected_output=(
            "A well-written legal brief in markdown format, "
            "ready for submission to the legal team."
        ),
        agent=legal_writer_agent
    )

# Initialize the crew with agents and tasks
crew = Crew(
    agents=[legal_researcher_agent, legal_writer_agent],
    tasks=[conduct_legal_research, draft_legal_brief],
    verbose=2
)

# Start the workflow with a specific input
result = crew.kickoff(inputs={"topic": "Employment Law"})

```

▼ Advanced

1. MarkdownTools.py file

```

import os
import sys
from langchain.tools import tool
from pymarkdown.api import PyMarkdownApi, PyMarkdownApiException

@tool("markdown_validation_tool")
def markdown_validation_tool(file_path: str) -> str:
    print("\n\nValidating Markdown syntax...\n\n" + file_path)

    scan_result = None
    try:
        if not (os.path.exists(file_path)):
            return "Could not validate file. The provided file path does not exist."

        scan_result = PyMarkdownApi().scan_path(file_path)
        results = str(scan_result)
        return results # Return the reviewed document
    except PyMarkdownApiException as this_exception:
        print(f"API Exception: {this_exception}", file=sys.stderr)
        return f"API Exception: {str(this_exception)}"

```

2. `.env` file

```

# Using OpenAI's API
# OPENAI_API_KEY="sk-..."
# MODEL_NAME="gpt-3.5-turbo"

# Using Ollama
MODEL_NAME='llama3'
OPENAI_API_BASE_URL="http://localhost:11434/v1"
OPENAI_API_KEY='ollama'

```

3. main.py file


```

tools=[markdown_validation_tool],
llm=defalut_llm)

# Define Tasks Using Crew Tools
syntax_review_task = Task(description=f"""
    Use the markdown_validation_tool to review
    the file(s) at this path: {filename}

    Be sure to pass only the file path to the m
    Use the following format to call the markdo
    Do I need to use a tool? Yes
    Action: markdown_validation_tool
    Action Input: {filename}

    Get the validation results from the tool
    and then summarize it into a list of change
    the developer should make to the document.
    DO NOT recommend ways to update the document
    DO NOT change any of the content of the docu
    add content to it. It is critical to your ta
    only respond with a list of changes.

    If you already know the answer or if you do
    to use a tool, return it as your Final Answer
    agent=general_agent)

updated_markdown = syntax_review_task.execute()

return updated_markdown

# If called directly from the command line take the fir
if __name__ == "__main__":
    if len(sys.argv) > 1:
        filename = sys.argv[1]

```

```
processed_document = process_markdown_document(1
print(processed_document)
```

4. pyproject.toml

```
[tool.poetry]
name = "markdown-validation-crew"
version = "0.1.0"
description = ""
authors = ["ITLackey <itlackey@gmail.com>"]

[tool.poetry.dependencies]
python = ">=3.10.0,<3.12"
crewai = "^0.11.0"
python-dotenv = "1.0.0"
markdown = "3.4.3"
pymarkdownlint = "0.9.15"

[tool.pyright]
# https://github.com/microsoft/pyright/blob/main/docs/configuration.md#command-line-arguments
useLibraryCodeForTypes = true
exclude = [".cache"]

[tool.ruff]
# https://beta.ruff.rs/docs/configuration/#command-line-arguments
select = ['E', 'W', 'F', 'I', 'B', 'C4', 'ARG', 'SIM']
ignore = ['W291', 'W292', 'W293']

[build-system]
requires = ["poetry-core>=1.0.0"]
build-backend = "poetry.core.masonry.api"
```

▼ Process of Running

- **Configure Environment:** Copy ``.env.example`` and set up the environment variables the model, endpoint url, and api key.

- **Install Dependencies:** Run `poetry install --no-root`.
- **Running the Script:** Execute `python main.py <path to markdown file>`. The script will leverage the CrewAI framework to process the specified file and return a list of changes.
 - **Execute the Script:** Run `python main.py README.md`

▼ Rags for PDF

Library

```
pip install langchain beautifulsoup4 chromadb gradio ollama
```

Code

```
import gradio as gr
import bs4
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.document_loaders import WebBaseLoader
from langchain_community.vectorstores import Chroma
from langchain_community.embeddings import OllamaEmbeddings
import ollama

# Function to load, split, and retrieve documents
def load_and_retrieve_docs(url):
    loader = WebBaseLoader(
        web_paths=(url,),
        bs_kwargs=dict()
    )
    docs = loader.load()
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000,
    splits = text_splitter.split_documents(docs)
    embeddings = OllamaEmbeddings(model="mistral")
```

```

        vectorstore = Chroma.from_documents(documents=splits, embedding=embedding)
        return vectorstore.as_retriever()

# Function to format documents
def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)

# Function that defines the RAG chain
def rag_chain(url, question):
    retriever = load_and_retrieve_docs(url)
    retrieved_docs = retriever.invoke(question)
    formatted_context = format_docs(retrieved_docs)
    formatted_prompt = f"Question: {question}\n\nContext: {formatted_context}"
    response = ollama.chat(model='mistral', messages=[{'role': 'user', 'content': formatted_prompt}])
    return response['message']['content']

# Gradio interface
iface = gr.Interface(
    fn=rag_chain,
    inputs=["text", "text"],
    outputs="text",
    title="RAG Chain Question Answering",
    description="Enter a URL and a query to get answers from the documents"
)

# Launch the app
iface.launch()

```

Output

RAG Chain Question Answering

Enter a URL and a query to get answers from the RAG chain.

url

https://docs.python-guide.org/scenarios/scrape/index.html

question

summarize it

Clear

Submit

output

processing | 63.6s

Flag