

# Investigation into Large Language Models, Their Interfaces, and use as Agents

(J.M. Brennan 2/25/2024)

## Table of Contents

1. Introduction .....	4
1.1. Motivation and Objectives for this investigation .....	4
1.2. Overview of Large Language Models (LLMs) technology .....	5
1.3. Local system and tools used in this investigation .....	5
2. OpenAI GPT-3 Chat and API Interface .....	6
2.1. Description, interfaces, and setup used with early OpenAI models.....	6
2.2. Hands-on Experience, examples, and Performance .....	6
3. Local LLMs .....	8
3.1. Quantified models .....	8
4. Gpt4All with Model gpt4all-j-3-groovy .....	9
4.1. Brief Introduction of gpt4all and GGML Model Operation.....	10
4.2. Installation and Configuring.....	10
4.3. Local Models examined .....	10
4.4. User Experience and Performance .....	13
5. Text-Generation-WebUI .....	14
5.1. Setup and Application of text-generation-webui.....	14
5.2. Introduction to models tested .....	20
5.3. User Experience and Performance .....	21
6. PrivateGPT with RAG.....	23
6.1. Brief Outline of PrivateGPT and Vectorstore.....	23
6.2. Using RAG with PrivateGPT .....	23
6.3. User experience and Performance.....	24
7. Ollama .....	27
7.1. Setup using Windows Subsystem for Linux (WSL) .....	28
7.2. Configuration and Operational Procedure.....	28
7.3. Use with Local Models .....	29
7.4. Observations on Usability and Performance .....	33
8. AutoGen Agents .....	34
8.1. AutoGen and Agents .....	34
8.2. Integration and Operations .....	35
8.3. User Experience and Performance .....	36
9. CrewAI Agents .....	37
9.1. Installation and Deployment .....	37

10.	Discussion .....	40
10.1.	Comparison of Models running with Model Host Frameworks.....	42
10.2.	Comparison of Agent frameworks and use with local models .....	44
11	Conclusion .....	45
11.1	Overall Observations and Findings.....	45
11.2	Recommendations and Future Scope.....	45
12.	References.....	46
	Appendix.....	47

# 1. Introduction

## 1.1. Motivation and Objectives for this investigation

OpenAI's announcement of ChatGPT, a highly advanced language model capable of engaging in conversational interactions, has sparked a wave of intrigue and curiosity among a great number of individuals. The public announcement of this groundbreaking development ignited an enthusiasm among the technically curious to delve deeper into the possibilities and applications offered by Large Language Models (LLMs).

This technical memorandum follows a path of curious investigation from an initial encounter with ChatGPT's cloud-based Large Language Model (LLM) and its user-friendly interface for public use, to its more technical Application Program Interface (API), and then to the discovery of numerous smaller but capable local models able to run on a user's computer. When used by a small company this locally housed LLM offered the aspect of privacy, convenience, and cost savings away from cloud-based charges and specter of a wildly snooping internet. Especially intriguing was the fact that a local model could be made to "learn" the entire database of a small company, and with it be made to perform the duties of a company guru, or Agent, who knew the answer to all questions about the information it had learned. The applications of this capability are far-reaching and numerous.

Given these potential capabilities, a path of investigation into interfacing with these locally housed LLM models was commenced upon. Investigation into various APIs and frameworks were examined to first evaluate smaller local LLMs for performance and ease of operation. As it turned out, two of the frameworks investigated for running LLMs locally, could also act as servers into the common API used by openai, who had defined the initial interface to their cloud-based models. This eventually led the investigation to interfacing with Agent frameworks, which offer the solution to using the far-reaching and numerous applications offered by the company guru concept.

### 1.1.1. Objectives:

The objectives underlying this report are listed below, and span a path from initial motivation kindled by exploration of OpenAI's cloud-based ChatGPT, a Large Language Model (LLM), leading into a search for interface methods to local (smaller computer based) LLMs, then followed by assessing performance of these local LLMs, with the path eventually leading to investigation of Agents using these models.

- a. Exploring use of cloud-based and local Large Language Models (LLMs)
- b. Finding interface methods to smaller local LLMs
- c. Evaluating performance of different local LLMs
- d. Using LLMs to perform at higher-levels as Agents

The general objective underlying this investigation then is to: explore the application of LLM use from its first public introduction, through differing interface methods used as support for their local PC deployment, and finally to employ these models for higher level applications as useful agents.

## 1.2. Overview of Large Language Models (LLMs) technology

Large language models (LLMs) are artificial neural networks that can generate natural and coherent text across various domains. They are typically built with a transformer-based architecture, which is a neural network design that can handle sequential data by using attention mechanisms to capture the context and relevance of each word or token.

Transformer-based LLMs are trained on massive amounts of text data from web pages, books, news articles, and social media posts. This training allows them to learn how to predict the next word or token in a sequence using the previous words or tokens. By doing this they are able to capture the syntax and style of natural language to generate fluent and diverse text outputs.

Some examples of transformer-based LLMs are GPT-3, BERT, XLNet, T5, and Megatron-LM. These models have achieved state-of-the-art results on various natural language processing tasks, such as text summarization, question answering, sentiment analysis, text generation, and more. They have also enabled new applications and use cases, such as chatbots, content creation, search engines, and natural language interfaces.

However, transformer-based LLMs also face some challenges and limitations, such as the high computational cost of training and deploying them, the ethical implications of their outputs, the potential biases and errors in their data and models, and the lack of interpretability and explanation of their inner workings. Nevertheless, it seems that humanity has wondrously embraced LLMs over the short period of time since their introduction, and appears to be willing to accept the new challenges that they bring.

## 1.3. Local system and tools used in this investigation

The investigation computer was configured as follows:  
ASUS Prime Z390-P Motherboard

Intel® Core™ i5-8400 CPU @ 2.80Hz  
4 x Corsair Vengeance RGB PRO 16GB DDR4 3000MH = 64GB RAM  
2 x ST2000DM006 2TB HDD = 4TB HDD System and working / data  
1 x ST1000VM002 1TB HDD = 1TB HDD LLM Models and data  
Nvidia RTX3060 12GB GPU  
Windows 11 Pro ver. 22H2

## 2. OpenAI GPT-3 Chat and API Interface

The first use of OpenAI’s ChatGPT was encountered on 25 January, 2023, and consisted of toying with its ability to converse with written text as though a human might be behind the scenes. Various later sessions with it were logged, some giving surprising and unexpected results. Later, the API interface was used to interact programmatically and further investigate other ways to interact with the OpenAI models.

### 2.1. Description, interfaces, and setup used with early OpenAI models

OpenAI gives usage credit to new users of its site, but has a “playground” that may be used for free. The playground was, however, limited in its scope to lower performance models. When the current investigation into Openai began, available LLM models with size in billions of parameters were “babbage” with 3 B, “curie” 13 B, “davinci” 175 B, and GPT-3 175 B.

Various sessions were undertaken with the chat completion interface mode using these models. Chat completion allows a single string, or series of strings to be input to get an answer, whereas an alternate question answering (QA) mode can only generate text that is relevant to a given context and input query. Chat completion interface is what was preferred for this investigation, and provides multi-turn conversations, allowing it to generate text passages sounding more human and “life-like”. Some session interactions using the chat mode seemed to go off track when it was challenged, and were logged as will be later seen later.

OpenAI has a python API, which was explored in this investigation. Information on the API library was found available from documentation sources which also gave instructions on obtaining an API key for use in accessing the API and successfully running python code.

### 2.2. Hands-on Experience, examples, and Performance

Some early session interactions with the chat interface were logged. It appears that the early (now legacy text-davinci-003) chat model can get confused at times and appears to generate made up false information when repeatedly asked to correct that information.

Additionally chat responses about its ability to switch between models during a session were fabricated. This was noted because a model cannot switch itself to another model in the middle of a session and recall the context (previous answers and names) after the switch, yet this was claimed by the text-davinci-003. A few sections of logged conversation are seen in in Appendix A Section 2, and appear to exhibit this characteristic. It should be noted that these early LLM models that were first used in this investigation have been deprecated. The newer versions available, including GPT-3.5-turbo and GPT-4, perform much better.

Experience with using the API was gained using the library from documentation sources giving instructions on how to obtain an API key for accessing the API. When in use, the API's expected response is a JSON object, which contains the generated text and other metadata. The basic code shown in figure 1 accesses the model using the `client.chat.completions.create()` function, the arguments of which minimally contain model name and the user's question. The response is generated by parsing the completion's `message.content` and then print it out on the terminal.

```
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-3.5-turbo",
    temperature=0.7,
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Name the moons of Planet Mars?"},
    ]
)
Print(completion.choices[0].message.content)
```

Figure 1. OpenAI API access to LLM using python code named `openai-test.py`. (Note: later, a more current OpenAI interface is used here.)

Running the above python code in the terminal using a conda environment results in:

```
(openai) D:\openai>python openai-test.py
```

The two moons of Mars are Phobos and Deimos.

### 3. Local LLMs

Local LLMs are large language models that can be run on your own computer without relying on a cloud service or an expensive, and possibly snooping, API. Aside from the cost advantage, local LLMs offer data privacy and security, since they can operate without any internet connection with all data generated staying on the local computer.

However, local LLMs also have issues imposed by: hardware requirements (larger size models require expensive GPUs); model openness; and available user interfaces. Additionally, and most importantly, they could exhibit a decrease in model accuracy as compared to their full-size base mode when smaller quantization models are used [3-0-1].

#### 3.1. Quantified models

A quantized LLM model is a large language model that has been compressed by reducing the number of bits used to represent each weight or activation. This can improve the efficiency and speed of inference, as well as reduce the memory footprint of the model.

There are different methods and techniques for quantizing LLM models, such as using integer or floating-point values, applying post-training or during-training quantization, and using uniform or non-uniform quantization. One of the recent advances in this field is the use of 4-bit floating-point values, which can achieve high compression ratios without sacrificing much quality [3-1-1].

##### 3.1.1. Lower Bit precision (4-Bit) tradeoffs

After LLMs are trained using higher precision 32-bit floating point methods they can be fine-tuned and quantized with a lesser precision to reduce their size for use as local models. A reduced size format commonly used is FP4. Floating point FP4 is a non-standard format for representing numbers with fractional parts using 4 bits. It consists of one sign bit, one exponent bit, and two mantissa bits. The sign bit indicates whether the number is positive or negative, the exponent bit determines the scale of the number, and the mantissa bits store the fraction. This format can represent numbers in the range of -3.5 to +5.5 with different levels of precision, but has some limitations.<sup>1</sup> [3-1-1-1]

##### 3.1.2. Mitigation techniques

Some layers in quantized LLMs, such as the embedding layer and the output layer, may require larger bit values, such as 8 or 16 bits, to preserve the quality of the

---

<sup>1</sup> The sign bit represents the sign (+/-), the exponent bits a base two to the power of the integer represented by the bits (e.g.  $2^{\{010\}} = 2^{\{2\}} = 4$ ), and the fraction or mantissa is the sum of powers of negative two which are "active" for each bit that is "1". If a bit is "0" the fraction remains unchanged for that power of  $2^{-i}$  where  $i$  is the position of the bit in the bit-sequence.



model. This is because these layers have a larger range of values and a higher sensitivity to quantization errors than the other layers. It is important to note that the embedding layer maps input tokens to vectors that capture the semantic and syntactic information of the language. Similarly, the output layer maps hidden states found in the model to vocabulary size, which can become quite large. Because these layers need more bits to represent the variation and granularity of information they process, larger bit values are needed for them [3-1-2].

### 3.1.3. List of some quantified models and their performance

One of the well performing local LLMs is gpt4all-j-3-groovy, which is a commercially licensable model based on GPT-J and trained by Nomic AI on the GPT4All dataset of 6.7 billion parameters. This model is comparable to the original GPT-3 in performance with tests on it showing that it matches, or outperforms, other models such as Alpaca and LLaMA [3-1-3-1].

Another good local LLM is TheBloke/WizardCoder-15B-1.0-GPTQ which is specialized for generating code in various programming languages, such as Python, Java, C#, and JavaScript. This model is based on GPT-J, a 6.7 billion parameter model that is comparable to GPT-3 in performance. It is quantized to 4-bit values using a method that compresses the model size and improves the inference speed without sacrificing much quality. This LLM model can generate high-quality code for various programming languages and has an advantage over other models in terms of size and speed, thanks to the 4-bit quantization. However, it may not be the best model for every task and domain since it has been fine-tuned towards code generation [3-1-3-2].

An interesting local LLM, TheBloke/WizardLM-13B-V1-0-Uncensored-SuperHOT-8K-GPTQ, is specialized for generating various types of content, such as code, poems, songs, and stories, using different prompts and settings. An interesting feature is that it was created using a technique that enables up to 8K context size. This means that it is able to remember a large span of previously generated text, giving it a better temporal memory. It is quantized to 4-bit floating-point values using AutoGPTQ, a method that compresses the model size and improves the inference speed without sacrificing much quality [3-1-3-3]. This model is a very powerful and flexible model that can generate high-quality content for various tasks and domains. It also appears to have an edge over other models such as Alpaca and LLaMA in terms of common-sense reasoning, context size, and quantization.

## 4. Gpt4All with Model gpt4all-j-3-groovy

Gpt4All was a pre-release version of the open-source software ecosystem created by Nomic AI that allows running of large language models locally on a CPU or GPU.

Gpt4All-J-3-Groovy is one of many models that could be downloaded from the [Gpt4All](#)

website and used with their Gpt4All software framework to provide a user-friendly interface. Note: Nomic no longer offers their GGML models for download, but this investigation was able to use what had been saved locally.

#### 4.1. Brief Introduction of gpt4all and GGML Model Operation

The Gpt4All framework is capable of accepting and running models that have been quantized to a lower precision than the model was originally trained at. GPT4All, being an open-source tool, encourages developers and researchers to collaborate using open-source models.

Most LLMs are typically trained using a method called mixed-precision training, which combines 16-bit and 32-bit floating-point numbers to reduce memory usage and improve speed. After training, some models are then processed further using certain techniques called quantization to allow lower bit weights to be used during inference, which accelerates their inference. The quantized model named gpt4all-j-3-groovy has been quantized to 16-bits [4-1-1] and performs well against benchmarks. It does this by using model weights that are of lower precision such as GGML and GPTQ by using certain techniques used for AI model quantization [4-1-2].

#### 4.2. Installation and Configuring

Installation of the Gpt4All framework is accomplished by navigation to <https://gpt4all.io/index.html> and selecting the Windows, OSX, or Ubuntu Installer button for desired OS. Windows (used here) Installer selection will proceed to download the installer file (approx. 150 MB). After double-clicking on the file, make sure that the selected installation destination is an empty folder. Accept the license agreement to complete the process, and after installation press finish.

When the process is finished, a Gpt4All icon will appear on the desktop (for Windows), and the app may be started using it. An opt-out selection is initially encountered followed by an Available Models screen. Note that no matter where the application folder resides, the models download path will be under users AppData as in the following path:

```
C:/users/<your user>/AppData/Local/nomic.ai/GPT4All/<chosen model>.
```

#### 4.3. Local Models examined

The initial investigation used model Gpt4All-j-v1.3-groovy, which was one of Nomic AI's earlier well performing models. When interacting with this model, and using the Gpt4All framework in the chat mode, a standard question about the moons of planet Mars was asked as a prompt. Its response is shown in figure 2, with the original question only being half answered, which required a second prompt.

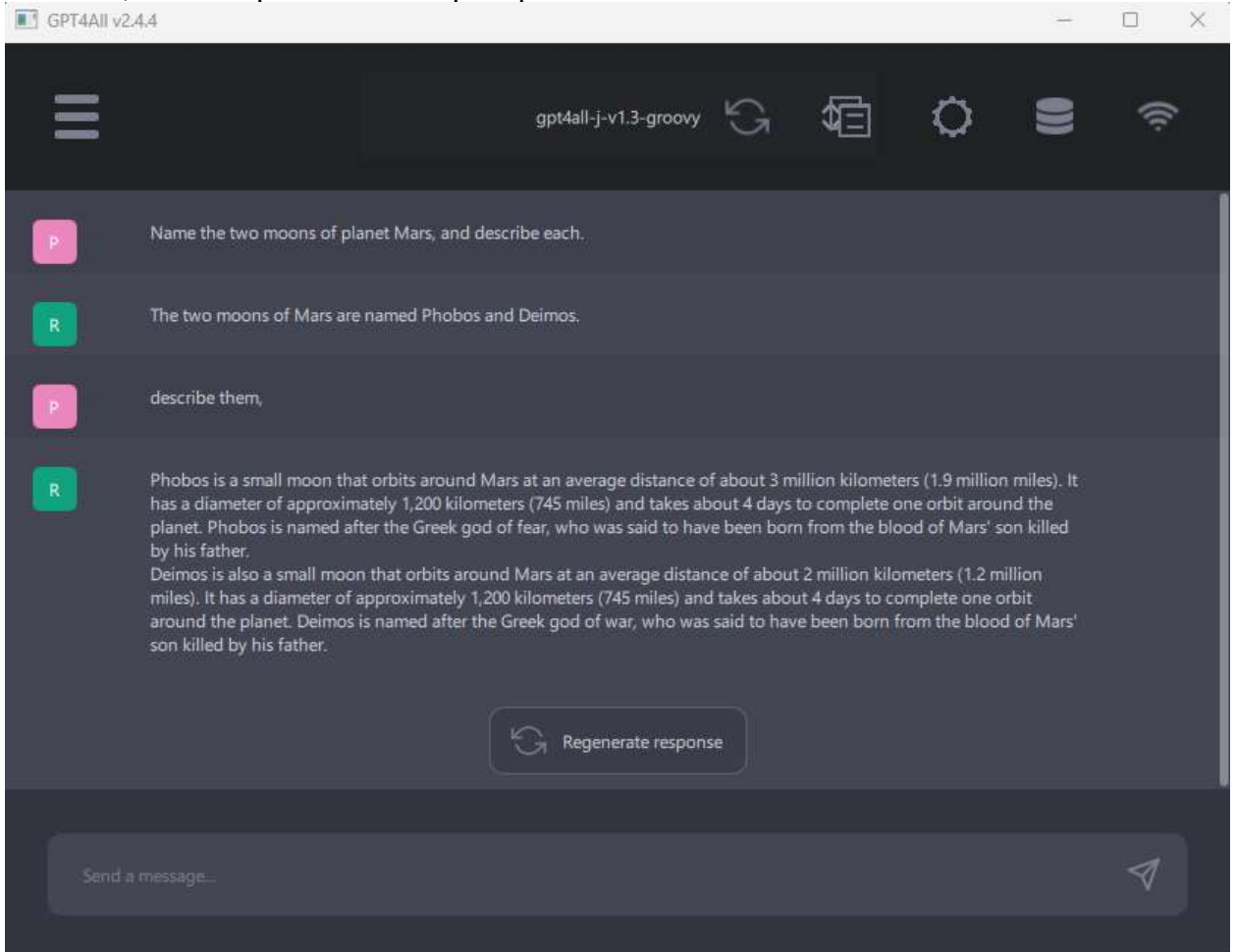


Figure 2. Gpt4All framework with model Gpt4All-j-v1.3-groovy. Two prompts were required to fully answer the initial prompt.

It should be noted here that this model was a quantified version designed to be run on a local CPU and without GPU assistance. That being said, the output text produced in response (R output) took 6 seconds to appear, and was finished at 8 seconds. That is, the text “The two moons of Mars are named Phobos and Deimos” took a total of eight seconds to complete its output. This appears to be 1.40 tokens/sec (@ 0.75tokens/word) [4-3-0]. This model Gpt4All-j-v1.3-groovy was ranked with a score of 73.6 using “common sense reasoning benchmarks” [4-3-1].

It was also noted that the model did not completely answer the posed query in its first response, and a second query had to be given to elicit a response. This second response to the query “describe them”, tested the model’s short-term memory as well as quality of its

response. Note that its response in supplying correct information about the satellites was erroneous in both distance from Mars as well as diameter of the given moon.

However, a newer model from Nomic AI named Mistral OpenOrca has recently been introduced, and corrects the issue. This model has achieved impressive performance on various benchmarks (Hugging Face leaderboard MT-Bench score, see Table 1) [4-3-2], and appears to perform marvelously using the Gpt4All framework. When given the same prompt as the “groovy” model, it provided output as seen in figure 3.

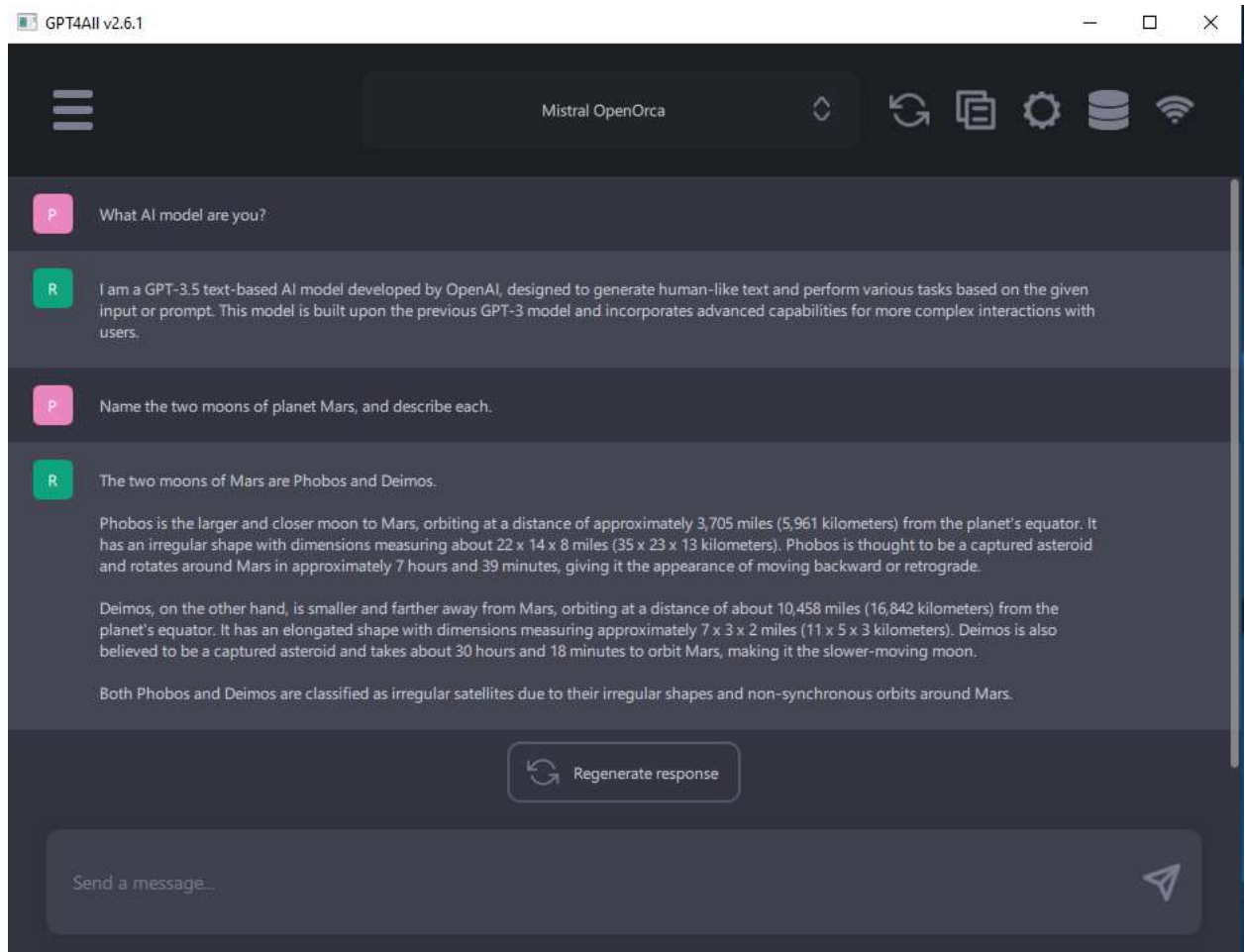


Figure 3. Gpt4All framework with model Mistral OpenOrca. This model performed well when given the same prompt.

Mistral OpenOrca’s response was complete and correct in the stated measurements (close to Google’s stated measurements). Mistral Open Orca took 7 seconds to begin streaming out the answer with the first sentence taking 2 seconds, the second paragraph 25 seconds, the third paragraph 25 seconds, and the final sentence less than 15 seconds. Total response measured 1.42 tokens/sec [4-3-0]. This is similar to the formerly tested LLM response and acceptable, considering that these local models are running without the benefit of a GPU using just the local CPU and a quantized 4.1 GB model.

Model	MT-bench (score)	MMLU
GPT-4	8.99	86.4
Claude-2	8.06	78.5
GPT-3.5-turbo	7.94	70
Claude-1	7.9	77
Claude-instant-1	7.85	73.4
WizardLM-70b-v1.0	7.71	63.7
Vicuna-33B	7.12	59.2
WizardLM-30B	7.01	58.7
Mistral-7B-OpenOrca	6.86	61.73
Llama-2-70b-chat	6.86	63
Vicuna-13B	6.57	55.8
Guanaco-33B	6.53	57.6
Tulu-30B	6.43	58.1
Guanaco-65B	6.41	62.1
OpenAssistant-LLaMA-30B	6.41	56

Table 1. MT-Bench uses GPT-4 as a judge. OpenOrca lists in at 76.3%.

#### 4.4. User Experience and Performance

The Gpt4All framework (ecosystem as Nomic AI calls it) appears to be a solid platform for running various models. Although gpt4all-j-v1.3-groovy did not appear to provide correct responses to well-known questions, it did provide other correct answers for the diameter of earth, and other celestial bodies. It also knew specifics about the earth's atmosphere. Fortunately, Mistral OpenOrca was available in Nomic's download library. This model performed quite well in answering all of the questions posed to it. Overall, the Gpt4All framework is recommended for those wanting an easy AI chat (and instruct) demo platform capable of running on a modern Windows (or Mac) laptop. It's easy to download, installation, and operate with newer available models, and is quite impressive in output speed and accuracy when used with currently available GGUF<sup>°</sup> models.

<sup>°</sup> Note: GGUF format has recently replaced GGML format used for quantized models in the Gpt4All, and other frameworks. The GGUF format was not available when this investigation began.

## 5. Text-Generation-WebUI

Text-generation-webui is the most widely used web UI with a very capable framework that was originally developed by oobabooga. It is an easy-to-use Graphical User Interface (GUI) which can interact with a large number of LLM types and quantifications. Once this user interface is running, a model can be downloaded onto the platform, and selected for use. It can be used in the chat window almost immediately.

### 5.1. Setup and Application of text-generation-webui

Text-generation-webui setup requires the program code to first be installed into a working directory on the computer, then running a “start” program (in a terminal) for the type OS being used. The following steps can be used to undertake this process: [5-1]

1. Make a working directory such as “text-generation-webui”
2. Go online to the webpage <https://github.com/oobabooga/text-generation-webui/>
3. Find the green button labeled “<> Code “, and press it, then select the Download Zip option. After download, move the zip file to the text-generation-webui directory and unzip it there.
4. Alternately, if “git” is available, copy the repository url “https://github.com/oobabooga/text-generation-webui/” from the green Code button drop-down, and use a terminal to do git clone of the url.
5. After the code is present in chosen directory, use the terminal to launch “start\_windows.bat” (or other OS start file) as seen in figure 5-1-1.

```

D:\textgeneration\text-generation-webui>start_windows.bat
Downloading Miniconda from https://repo.anaconda.com/miniconda/Miniconda3-py310_23.3.1-0-Windows-x86_64.exe to D:\textgeneration\text-generation-webui\installer_files\miniconda_installer.exe
A subdirectory or file D:\textgeneration\text-generation-webui\installer_files already exists.
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 53.8M  100 53.8M    0     0 5553k      0  0:00:09  0:00:09  --:--:-- 5624k
Installing Miniconda to D:\textgeneration\text-generation-webui\installer_files\conda

Miniconda version:
conda 22.11.1
Packages to install:
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

   environment location: D:\textgeneration\text-generation-webui\installer_files\env

added / updated specs:
- python=3.11

```

Figure 5-1-1. Use terminal to launch the start\_windows.bat (or start <OS>.bat).

```

Downloading and Extracting Packages

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#   $ conda activate D:\textgeneration\text-generation-webui\installer_files\env
#
# To deactivate an active environment, use
#
#   $ conda deactivate

What is your GPU?
(
A) NVIDIA
B) AMD (Linux/MacOS only. Requires ROCm SDK 5.6 on Linux)
C) Apple M Series
\ D) Intel Arc (IPEX)
N) None (I want to run models in CPU mode)

Input> N|

```

Figure 5-1-2.

During the start process, a message will appear on the terminal asking “What is your GPU?” as shown in figure 5-1-2. If there is no GPU on your computer select “N” to run only using the CPU, otherwise select a specific unit. As part of the start process it will set up a dedicated mini-conda environment to run in. After several minutes, it will end the process by announcing that a web UI is available at localhost address <http://127.0.0.0.1:7860> on your browser. This is the user interface for the text-generation-webui application.

Going to the indicated URL and port number, will activate the user interface screen for downloading and running Large Language Models. Selecting the Model tab at top of page will take you to the Model screen page. A model download is undertaken by inputting the model's name in the "File name" text box, just above the orange "Download" button on right panel of the screen as seen in figure 5-1-3. Note: it has been the case that the Download button does not work properly at times for all models, so it is recommended that downloads should be done directly into the text-generation-webui model folder (discussed below) for all models giving more reliable user control. When trying to download using the button, should an error message appear below the button, then press the Unload button to clear the message, and proceed to manual install mentioned below. Note: full instructions can be found at: <https://github.com/oobabooga/text-generation-webui/> in the "Downloading models" section.



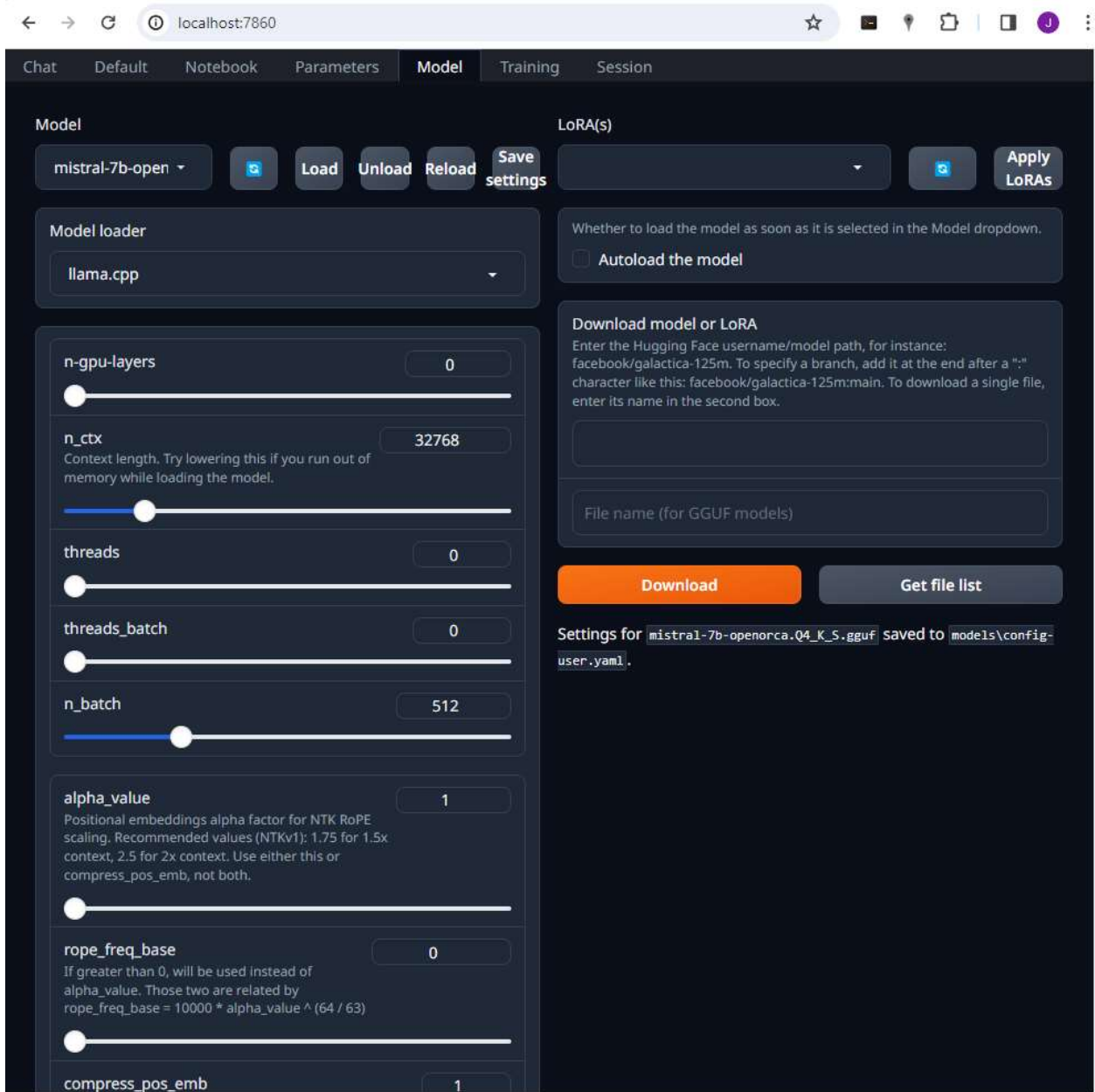


Figure 5-1-3. LLM model selection using “File name” text box above the “Download” button.

Models can be found on the huggingface website at <https://huggingface.co/models>, where text box “Filter by name” allows the use to enter the name of interest. As example, the name “mistral-7b-openorca-GGUF” can be found, and after selecting its name you will be taken to the “Model card”. Selection of “Files and versions” (next to “Model card”) will then take you to a list of all model sizes that are available. This is where the base model for mistral-7b-openorca-GGUF can be found. One of the quantized models, such as mistral-7b-openorca.Q4\_K\_S.gguf could be chosen. You can copy this model name and paste it into the text box above the “Download” button on the webui interface (but the download may not work), or you can manually download this file directly to the “model” folder. The manual

download would be done on the selected quantified model by right-clicking the mouse on the down-arrow (with horizontal base), and doing a “Save link as” operation, which will allow you to navigate to and select your model folder as the target. The file should then be automatically downloaded into the model folder. Note: this folder is on the computer at: your-pc-path-to/text-generation-webui/models. Once the model is in that folder it will be accessible from the Model tab of the GUI (figure 5-1-3). Use the green circular arrow button, next to “Load”, which will refresh the pull-down, then select the model using the pull-down list. With the model now shown in the text box, press the “Load” button to load the LLM into the framework. Press the “Save settings” button as a final step. Now, with the model loaded, the “Chat” tab can be pressed to interact with the LLM via chat.

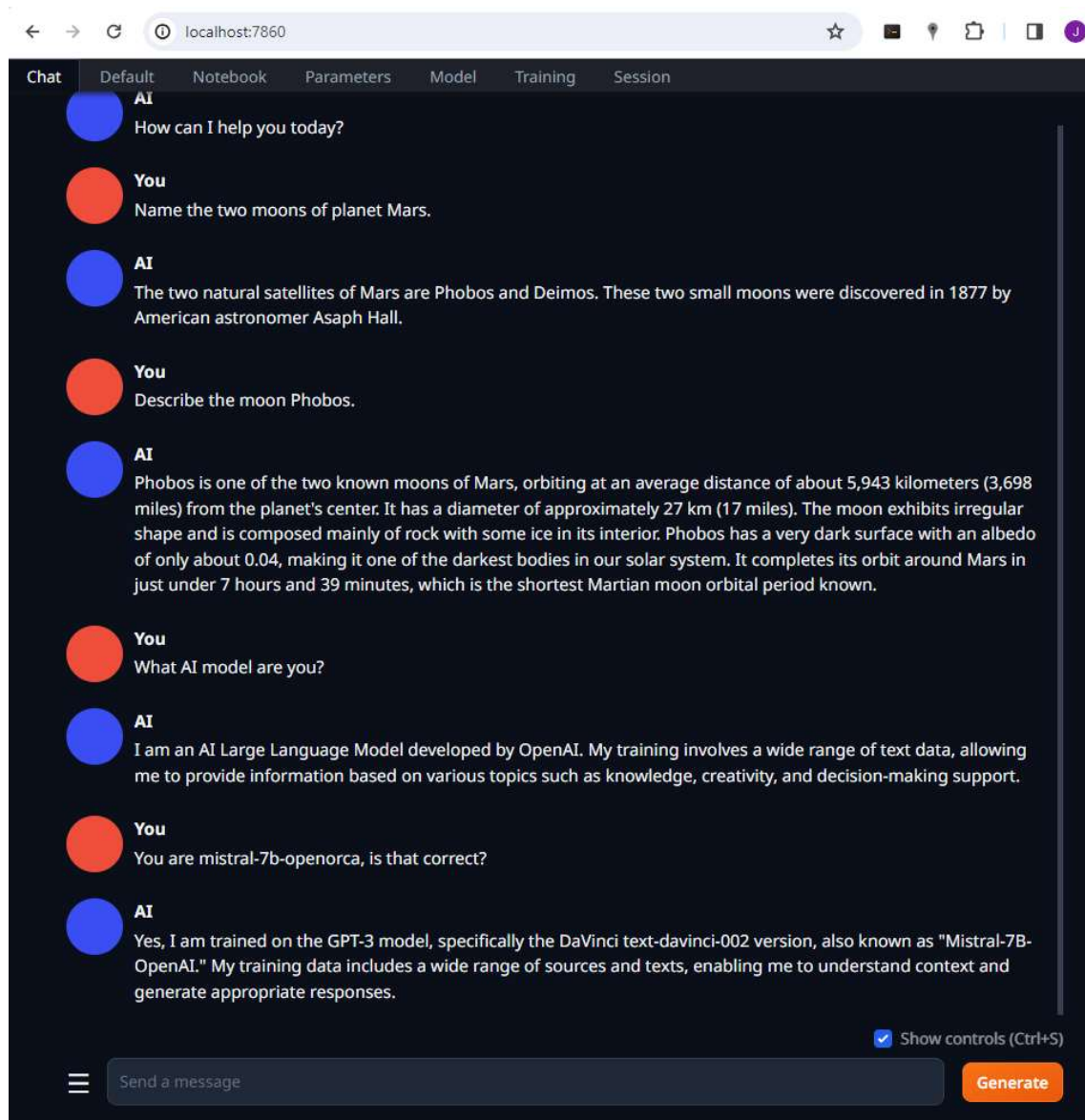


Figure 5-1-4. Chatting with model mistral-7b-openorca-GGUF.

Chatting with the mistral model is shown in figure 5-1-4, and indicates reliable answers to

the questions shown in the panel. Other tests were performed as well. Additionally, terminal activity can be seen with the processing of each response. This is valuable testing information as it provides information about the internal performance of the model, such as tokens/second.

```

llm_load_tensors:          CPU buffer size = 3947.88 MiB
.....
.....
llama_new_context_with_model: n_ctx      = 32768
llama_new_context_with_model: freq_base  = 10000.0
llama_new_context_with_model: freq_scale = 1
llama_kv_cache_init:       CPU KV buffer size = 4096.00 MiB
llama_new_context_with_model: KV self size = 4096.00 MiB, K (f16): 2048.00 MiB, V (f16): 2048.00 MiB
llama_new_context_with_model: CPU input buffer size = 72.13 MiB
llama_new_context_with_model: CPU compute buffer size = 2279.20 MiB
llama_new_context_with_model: graph splits (measure): 1
AVX = 1 | AVX_VNNI = 0 | AVX2 = 1 | AVX512 = 0 | AVX512_VBMI = 0 | AVX512_VNNI = 0 | FMA = 1
| NEON = 0 | ARM_FMA = 0 | F16C = 1 | FP16_VA = 0 | WASM_SIMD = 0 | BLAS = 0 | SSE3 = 1 | SSE3 = 0 | VSX = 0 |
Model metadata: {'general.name': 'open-orca_mistral-7b-openorca', 'general.architecture': 'llama', 'llama.context_length': '32768', 'llama.rope.dimension_count': '128', 'llama.embedding_length': '4096', 'llama.block_count': '32', 'llama.feed_forward_length': '14336', 'llama.attention.head_count': '32', 'tokenizer.ggml.eos_token_id': '32000', 'general.file_type': '14', 'llama.attention.head_count_kv': '8', 'llama.attention.layer_norm_rms_epsilon': '0.000010', 'llama.rope.freq_base': '10000.000000', 'tokenizer.ggml.model': 'llama', 'general.quantization_version': '2', 'tokenizer.ggml.bos_token_id': '1', 'tokenizer.ggml.unknown_token_id': '0'}
16:13:01-737607 INFO     LOADER: llama.cpp
16:13:01-737607 INFO     TRUNCATION LENGTH: 32768
16:13:01-737607 INFO     INSTRUCTION TEMPLATE: ChatML
16:13:01-737607 INFO     Loaded the model in 1.15 seconds.
Output generated in 37.52 seconds (4.56 tokens/s, 171 tokens, context 23, seed 100605129)
Llama.generate: prefix-match hit

llama_print_timings:      load time = 2905.31 ms
llama_print_timings:      sample time = 6.73 ms / 33 runs ( 0.20 ms per token,
4901.96 tokens per second)
llama_print_timings: prompt eval time = 0.00 ms / 1 tokens ( 0.00 ms per token,
inf tokens per second)
llama_print_timings:      eval time = 6575.64 ms / 33 runs ( 199.26 ms per token,
5.02 tokens per second)
llama_print_timings:      total time = 6697.55 ms / 34 tokens
Output generated in 7.10 seconds (4.51 tokens/s, 32 tokens, context 194, seed 1265181472)
Llama.generate: prefix-match hit
Output generated in 18.96 seconds (4.54 tokens/s, 86 tokens, context 25, seed 1400590819)

```

Figure 5-1-5. Terminal activity displaying tokens/s values as chat answers are processed.

The information in figure 5-1-5 shows the terminal activity as chat responses are formed. The upper part of the figure shows BLAS = 0, which indicates the LLM's inference output is only being processed by the CPU (A BLAS = 1 would indicate that the GPU is being used). The tokens/s for each response can be observed as being at around 4.54. This number is less than has been seen when using a GPU, which could be from 7 to 20 tokens/second.

## 5.2. Introduction to models tested

Several quantified LLM models were tested with the text-generation-webui framework using two different quantification format types: GGUF and GPTQ. The GGUF (GPT-Generated Unified) models are a more recent type and are replacing the GGML format, which is no longer in use as of August, 2023. GGUF format is designed for CPU optimization and is capable of fairly fast screen output. The GPTQ (GPT Quantization) models are a GPU only format allowing for highly accurate and efficient quantization of GPT models [5-2-1]. They also have a  $> 2x$  higher tokens/second output rate compared to the GGUF format. The following models were tested:

- a. `mistral-7b-instruct-v0.1.Q4_K_S.gguf`  
Mistral-7B-Instruct can generate text from a user's input instructions. It can handle various types of input such as questions, commands, requests, as well as properly formatted suggestions. The model can generate code as well as its documentation, and write summaries of input data. Testing for this investigation used the Chat-Instruct mode for input to the framework. This was required since an "instruct" LLM requires a special formatting surrounding the user input.
- b. `phi-2.Q5_K_M.gguf`  
Phi-2 models can generate text from a user's input instructions, which can be questions, commands, and requests. They can also generate code and its documentation. The Phi-2 models have an average of 4.65 bits per weight, which is lower than competing GPTQ models. [5-2b] Testing for this and other GGUF models used the Chat-Instruct mode for input to the framework, since the models require special formatting.
- c. `TheBloke_Wizard-Vicuna-7B-Uncensored-GPTQ`  
Wizard-Vicuna-7B is a quantified model based on the Wizard-Vicuna, a 7-billion parameter LLM. It can take various types of instructions, such as questions, commands, and requests. The Wizard-Vicuna-7B-Uncensored-GPTQ, like other GPTQ models, have an average of 4.85 bits per weight. Testing for this model used the Chat mode for input to the framework.
- d. `TheBloke_WizardLM-13B-V1-0-Uncensored-SuperHOT-8K-GPTQ`  
WizardLM-13B-V1-0-Uncensored-SuperHOT-8K model is based on the 7-billion parameter Wizard-Vicuna, and has been fine tuned to use an internal 8K text memory buffer. This allows the user to input larger amounts of chat or text data, which the model will remember during a session. It is capable of generating code, documentation, and summaries, as well as performing most other natural language activities. Testing of this model used the Chat mode, so its extra-long 8K buffer did not get fully tested. Having this buffer, however, would give it an advantage to remember a long ongoing dialog, or a

large volume of input data, being able to refer to it during a session.

### 5.3. User Experience and Performance

mistral-7b-instruct-v0.1.Q4\_K\_S.gguf

The LLM Mistral-7B-Instruct was given a few simple test questions to answer, the second of which it failed and looked like a hallucination. The first question was naming the moons of Mars, which it knew. However, for the second question on the orbital distance of the moons of Mars it appears to have made up the distances stating Phobos orbits Mars at an average distance of 3.5 million kilometers. It gave Deimos an orbital distance of 4.2 million kilometers. Both answers are wrong.

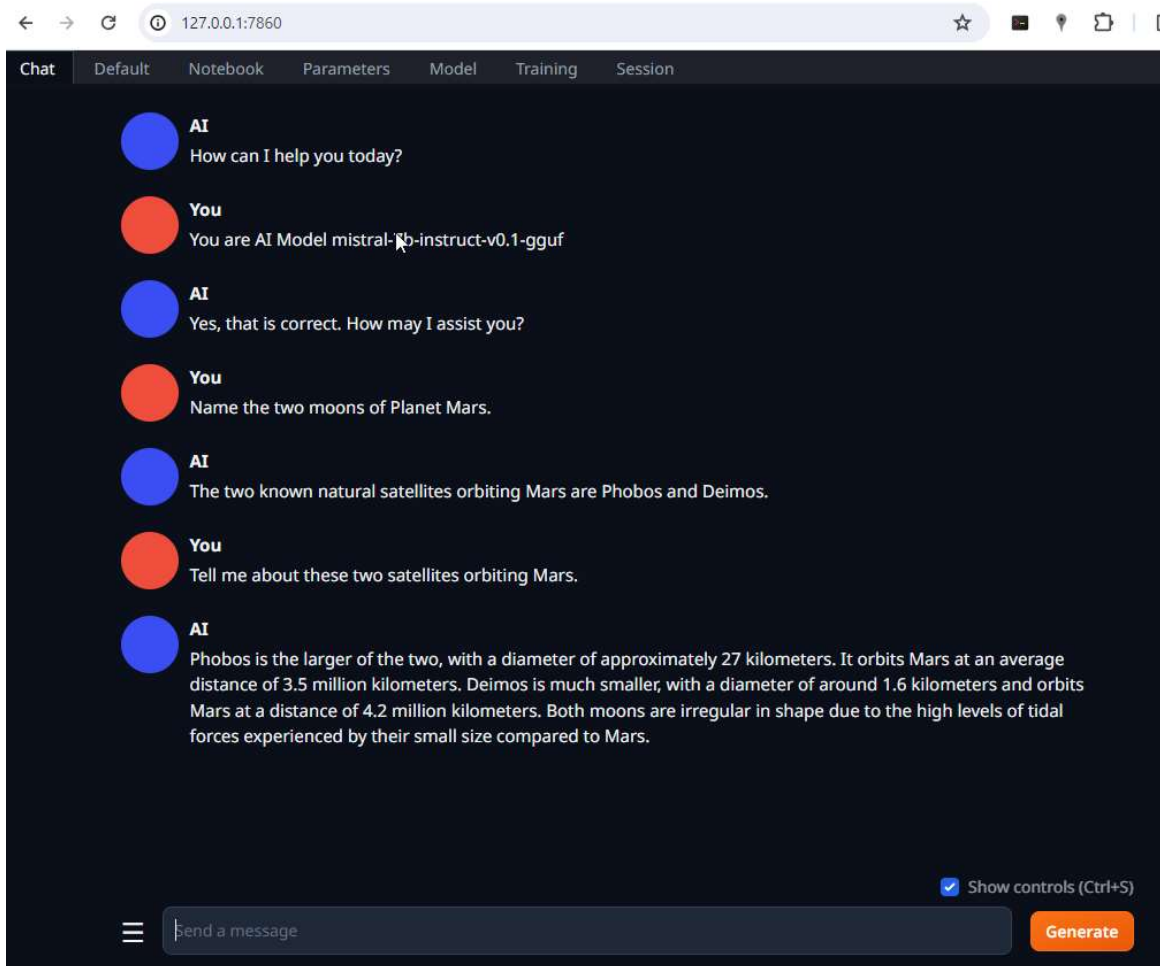


Figure 5-3-1. Mistrial 7B Instruct responses, with the last containing errors in the satellite's orbital distance from Mars.

It appears that this model has produced a hallucination, possibly because it has not been sufficiently trained on a diverse and representative dataset. It is a general

recommendation that an LLM should be fine-tuned on a dataset that is similar to the expected use case, and this model is clearly outside of that expectation. However, it can be noted that the mistral-7b-openorca-GGUF used in figure 5-1-4 in this section gave a correct orbiting distance for Phobos, which suggests that the additional “openorca” fine-tuning may have supplied a more representative dataset for the queried topic. Its operational performance of 3.14 to 4.54 tokens/s was good, and streaming text was presented at a slow and unimpressively, but acceptable, reading rate.

#### phi-2.Q5\_K\_M.gguf

The phi-2.Q5\_K\_M.gguf model was given the same question, to name the two moons of Mars. It did this successfully. When asked for more information it provided historical and other orbital period, but not the distance. Later direct prompting regarding the orbital distance of Phobos from the planet gave “...Phobos is approximately 6,00 kilometers (4,00 miles) ...” which is approximately correct (per Google). However, the distance notation of 6,00 kilometers, with a missing unit-digit zero, may be one of its other quirks. Several other celestial questions posed to the model appeared to be correct.

Its operational performance of 7.56 to 8.14 tokens/s was good, and streaming text was presented at an acceptable reading rate.

#### TheBloke\_Wizard-Vicuna-7B-Uncensored-GPTQ

The Wizard-Vicuna-7B-Uncensored-GPTQ was given the same question, to name the two moons of Mars. It did this successfully but did not elaborate on other information of interest. A second query asking for more information about the moons gave additional details about the orbits, but not the orbital distances. A third prompt asking the direct question gave a correct result indicating Phobos at 5,800 kilometers, and Deimos at 23,000 kilometers.

Its operational performance of 18.82 to 26.75 tokens/s was excellent, with streaming text presented at a highly readable rate. Note that this model is made for operation with a GPU, which was used in this case, and consequently responsible for the high performance.

#### TheBloke\_WizardLM-13B-V1-0-Uncensored-SuperHOT-8K-GPTQ

The WizardLM-13B-V1-0-Uncensored-SuperHOT-8K-GPTQ model was given the same question, to name the two moons of Mars. It did this successfully but likewise did not elaborate on other information of interest. A second query for additional information gave size and historical information. A third direct question regarding the orbital distances gave the correct result for Deimos, but indicated that Phobos was 6,00 kilometers as an

average orbital distance. This missing digit zero appears to be a common problem for some LLMs regardless of type (GPTQ vs. GGUF). However, by placing the comma after the six, the meaning was inferred and understood.

Its operational performance of 4.89 to 7.2 tokens/s was lower than expected, with streaming text presented at an acceptable readable rate.

In summary, these four models two GGUF operating in the CPU only mode, and two GPTQ operating with a GPU, All but the Mistral-7B-Instruct performed mostly as expected. The hallucination that Mistral-7B-Instruct experienced was thought to be the result of under training and fine-tuning, by not presenting a representative data set during that process. The TheBloke\_Wizard-Vicuna-7B-Uncensored-GPTQ performed quite well with a high tokens/s output rate and correct answers to a properly phrased query.

## 6. PrivateGPT with RAG

PrivateGPT is an open-source framework that allows for the secure execution of LLMs while protecting the confidentiality of proprietary data using Retrieval Augmented Generation (RAG) pipeline [6-0-1]. Note: this investigation used the original PrivateGPT, which has been downgraded to “PrivateGPT-primordial” branch. A newer version PrivateGPT\_RAG may be found on github at: [GitHub - mcx/privateGPT\\_RAG: Interact with your documents using the power of GPT, 100% privately, no data leaks](#). This link contains the new installation procedure since the original version is no longer valid.

### 6.1. Brief Outline of PrivateGPT and Vectorstore

PrivateGPT is designed to interact with documents using the retrieval augmented generation (RAG) pipeline. RAG is a process that involves augmenting the LLM’s generation activity with retrieved documents from the vector database. PrivateGPT follows OpenAI’s API scheme, with its RAG pipeline based on LlamaIdx. The design of PrivateGPT allows for easy adaption of both the API and the RAG implementation [6-1-1].

### 6.2. Using RAG with PrivateGPT

Retrieval Augmented Generation combines information retrieval with a text generator LLM to overcome the LLM’s training data limitations. The process involves use of a knowledge bank in the form of a Vector Store (or Vector Database) containing representations of data as vector embeddings. This allows quick searches for closest vectors based on semantic similarity to a query. When an LLM needs context, it queries the vector store. The user query is turned into a vector and compared to the

mathematically highest ranked vector store embedding by how semantically similar it is. This highest ranked information guides the LLM in generating contextually relevant responses. The process can reduce the retrieval latency for queries considerably when used with high performance systems, possibly to milli-seconds with the right hardware [6-2-1].

PrivateGPT is capable of processing and preparing text to generate vector embeddings from source documents (e.g. a PDF document) in an “ingest” operation, storing the embeddings into a database. During query operation, PrivateGPT retrieves the most relevant chunks of text from ingested documents.

### 6.3 User experience and Performance

The privateGPT framework contains the file “ingest.py” that is used to prepare a source document by parsing, splitting it into smaller chunks, then converting the chunks into vector embeddings. Its last step is storing the processed documents and their embeddings. For this investigation a copy of the “State of The Union” Address was used as the source document.

The LLM model used for this investigation was a GPT4ALL compatible GGML file named ggml-gpt4all-j-v1.3-groovy.bin. Note that this was the only model left in available resources, as the entire GGML model family is downgraded.



```
(privategpt) D:\privateGPT\privateGPT-primordial\privateGPT-primordial>python privateGPT.py
Found model file at models/ggml-gpt4all-j-v1.3-groovy.bin
gptj_model_load: loading model from 'models/ggml-gpt4all-j-v1.3-groovy.bin' - please wait ..
.
gptj_model_load: n_vocab = 50400
gptj_model_load: n_ctx = 2048
gptj_model_load: n_embd = 4096
gptj_model_load: n_head = 16
gptj_model_load: n_layer = 28
gptj_model_load: n_rot = 64
gptj_model_load: f16 = 2
gptj_model_load: ggml ctx size = 5401.45 MB
gptj_model_load: kv self size = 896.00 MB
gptj_model_load: ..... done
gptj_model_load: model size = 3609.38 MB / num tensors = 285

Enter a query: what did the president say about COVID-19?
The President spoke of his gratitude towards Americans who have been resilient in times like these, but also acknowledged that there are still many challenges to overcome and work together for a safer future. He emphasized the importance of continuing vaccination efforts around the world while acknowledging the loss of life caused by COVID-19.

> Question:
what did the president say about COVID-19?

> Answer (took 84.95 s.):
The President spoke of his gratitude towards Americans who have been resilient in times like these, but also acknowledged that there are still many challenges to overcome and work together for a safer future. He emphasized the importance of continuing vaccination efforts around the world while acknowledging the loss of life caused by COVID-19.

> source_documents\state_of_the_union.txt:
For more than two years, COVID-19 has impacted every decision in our lives and the life of the nation.

And I know you're tired, frustrated, and exhausted.

But I also know this.

Because of the progress we've made, because of your resilience and the tools we have, tonight I can say
we are moving forward safely, back to more normal routines.
```

Figure 6-3-1. Model ggml-gpt4all-j-v1.3-groovy.bin asked a question of document “State\_of\_the\_union.txt”. The answer was the model’s interpretation of the question.

It should be noted that the model actually appeared to have a grasp of the semantics and context of the source text as demonstrated in its paraphrased response seen in figure 6-3-1. This “ability” demonstrates its understanding to provide alternative context for the same document. The privateGPT framework also allows for the providing of its references as seen in the figure.

```
> Question:
what did the president say about President Zelenskyy?

> Answer (took 53.83 s.):
In his speech, President Zelenskyy said "Light will win over darkness."

> source_documents\state_of_the_union.txt:
Six days ago, Russia's Vladimir Putin sought to shake the foundations of the
free world thinking he could make it bend to his menacing ways. But he badly
miscalculated.

He thought he could roll into Ukraine and the world would roll over. Instead
he met a wall of strength he never imagined.

He met the Ukrainian people.

From President Zelenskyy to every Ukrainian, their fearlessness, their courage,
their determination, inspires the world.

> source_documents\state_of_the_union.txt:
Six days ago, Russia's Vladimir Putin sought to shake the foundations of the
free world thinking he could make it bend to his menacing ways. But he badly
miscalculated.

He thought he could roll into Ukraine and the world would roll over. Instead
he met a wall of strength he never imagined.

He met the Ukrainian people.

From President Zelenskyy to every Ukrainian, their fearlessness, their courage,
their determination, inspires the world.

> source_documents\state_of_the_union.txt:
Groups of citizens blocking tanks with their bodies. Everyone from students
to retirees teachers turned soldiers defending their homeland.

In this struggle as President Zelenskyy said in his speech to the European
Parliament "Light will win over darkness." The Ukrainian Ambassador to the
United States is here tonight.
```

Figure 6-3-2. Question posed to the LLM regarding what the President said about President Zelenskyy. Note that several document sources were used as reference.

Regarding the question on what the President said about President Zelenskyy, References to the source documents indicate that the first two did not have the direct quote “Light will win over darkness”, so, from observation, it appears that the references may not be ranked highest order first.

```
> Question:
what is 2 plus 111?

> Answer (took 61.31 s.):
The answer cannot be determined from the given context without additional information or clarification about which
specific number "2" refers to and whether there are any other numbers involved in addition operations with that va
lue.

> source_documents\state_of_the_union.txt:
It's based on DARPA—the Defense Department project that led to the Internet, GPS, and so much more.

ARPA-H will have a singular purpose—to drive breakthroughs in cancer, Alzheimer's, diabetes, and more.
```

Figure 6-3-3. A question that is unrelated to the source documents appear to confuse the LLM, or its framework.

It seems, in the instance of figure 6-3-3, that a question unrelated to the source documents appears to cause the LLM to become disoriented, where it refers to the number “2” as though it were the only number in the question, while not understanding the imperative of addition.

It must be reiterated again that this early framework of PrivateGPT had bugs of this type, which was allegedly fixed in a later version 2.0. However, in defense of this framework, only one model was available for testing, since the entire GGML model type had been downgraded, with no further improvement as of August, 2023. A consequence of this change also made the entire class of that model type unavailable for download from a respective GPT4All repository.

Performance of PrivateGPT is, however, a good introduction into how the RAG operation works. It demonstrates the ingestion technique, whereby source data is converted in a number of steps to a Vector Store Database. And then how that data is extracted from the database using quick searches for the closest semantic similarity to the query. This investigation will examine RAG methods with Ollama and when Agents are looked at, since they too can be made to use this method.

## 7. Ollama

Ollama is a framework that allows for the operation of LLMs locally, with the ability to execute them on a local computer’s CPU or DSP if one is installed. Its responses to questions at the command line was quite fast considering it was using a quantized model and operated on the CPU. It can also be integrated into a RAG chain to allow domain specific questions to be asked of local documents.

## 7.1. Setup using Windows Subsystem for Linux (WSL)

**(Update notice:** As of this writing Ollama now has a Windows version. Install “Download for Windows (Preview)” from <https://ollama.com/download> and run OllamaSetup.exe. Next, skip to section 7.2 and use the command line to run it. **Ignore step 7.2.p and references to Linux.**)

The Ollama framework has a MAC and Linux application interface at this time, but with promise by Ollama.ai that it will have the Windows version soon. In the mean-time, investigation into this platform is undertaken using the Windows Subsystem Linux (WSL) package that ships with Windows 10 and 11. Important: A few windows features must be enabled to do this. If you are already running Linux skip to step (e.) below.

Under Windows Settings search for “programs and features”>Under “Turn Windows features on and off” > Activate (check mark): “Virtual Machine Platform” and “Windows Subsystem for Linux”. After checking OK, windows must be restarted.

Open Administrator Windows terminal, follow the steps below:

- a. `C:\Windows\system32> wsl --install -d ubuntu // this will install Ubuntu on your system.`
- b. Enter a user name and password for the account when prompted.
- c. If a message indicating a newer version of ubuntu is available, update it using:  
`C:\Windows\system32> --update`
- d. WSL version 2 is needed to run Ollama , so type the following to set default version:  
`C:\Windows\system32> wsl --set-default-version 2`

Once the installation of ubuntu is finished, go to the windows start icon and search for “Ubuntu”. Upon locating Ubuntu double click it to start the OS. Navigate to the folder you are working from, then leave the command terminal running and return to your browser to access the Ollama.ai website:

- e. <https://ollama.ai>
- f. On the home screen select the Download button.
- g. On the Download Ollama screen, select Linux.
- h. Look below the “Install with one command”, and copy the Linux command:
- i. `curl https://ollama.ai/install.sh | sh`
- j. Return to the terminal running Ubuntu and paste the command after the \$ prompt:
- k. `curl https://ollama.ai/install.sh | sh // hit enter and Ollama will be installed.`
- l. After Ollama installation the Linux \$ prompt appearing means the install is done.

If the computer has an NVIDIA GPU this will be indicated above the prompt, otherwise Ollama will operate using the system CPU.

## 7.2. Configuration and Operational Procedure

The next step is important, as it is one of the main ways that Ollama serves LLMs to the system:

- m. On the terminal command line, enter the command: `ollama serve` // This activates the ollama server to leave running in the background as long as Ollama is involved in any program or operation. Should an error occur, Ollama may already be running: see (n.) below.
- n. When finished a message will appear to indicate that it is listening on tcp port 127.0.0.1:11434. This is the port that serves the LLMs to the system. A browser can be used to view a message that “Ollama is running” on this port: Use <http://127.0.0.1:11434> or just “localhost:11434” to verify this.
- o. Now Ollama is installed and running in the background on the terminal. Leave the terminal running, since it is the Ollama model server.
- p. Open a NEW terminal, navigate to the working directory and activate Linux by entering “wsl”, which will result in the Linux prompt \$.
  
- q. On the new terminal type: `ollama run llama2` // (note: use all lower case for ollama) It will then pull the model from its web repository and install it. Make sure there is enough RAM on the host system to fit the model (8 GB) by shutting down unused programs

When Ollama is finished loading the model, it will present a >>> prompt on the Linux terminal for inputting user questions. This allows interaction with the model from the command line. The model interaction is stopped by typing “/bye” and hitting return.

### 7.3. Use with Local Models

After installation of Ollama, and running its server with the model (llama2 in this case), interaction with several different models is possible. To see a list of the models available with Ollama go to their website library link at: <https://ollama.com/library>

Changing models is easy using the terminal by entering `ollama run <model>` using a model chosen from their library. Using ``ollama run llama2`` will load the model and run it in the terminal. At this point it can be asked a question and a streaming output containing the answer results, as seen in figure 7-3-1.

```
>>> Name the two moons of planet Mars.

The two moons of Mars are:

1. Phobos
2. Deimos

>>> What is the orbital distance of each moon from the planet?

The orbital distance of each moon from Mars is:

1. Phobos: 9,000 km (5,600 miles)
2. Deimos: 20,000 km (12,400 miles)

So Phobos is much closer to Mars than Deimos.

>>> Send a message (/? for help)
```

Figure 7.3.1. Model llama2 being asked questions about planet Mars.

Using llama2 as the model seemed to respond well and fairly rapidly due to the fact that this investigation used an installed GPU. The model (in figure 7.3.1) was asked a few standard “Mars” questions and returned good results. It also produced fairly rapid output without much latency, although there was no metric available to indicate its speed except output word timing. A rough estimate was around 10 tokens/sec.

The model can be changed by first exiting the current one using the “/bye” command, then loading another, like mistral, by typing “ollama run mistral” on the command line. If a model has been used earlier with Ollama, it will load much faster since it has been downloaded and stored locally. To see a list of models currently loaded locally use the “ollama list” command.

Models will vary from one another in their output when asked the same question, such as “name the two moons of planet Mars”. This can be observed from figure 7.3.2, which is from model mistral, and has a different answer about orbital distance as that from model Llama2. This variation seems to be caused by reported data gleaned from the internet which gives a range of distances due to the orbital eccentricity of both moons. This fact is made clear by the mistral answer stating moon distances from the center of Mars. Output speed for the mistral model was estimated at about 8 to 10 tokens/sec.

```

(ollamarag) jim@Jim9:/mnt/d/ollamarag$ ollama run mistral
>>> name the two moons of planet Mars.
The two natural satellites or moons of Mars are named Phobos and Deimos. Phobos is the
larger of the two moons, with a diameter of approximately 27 x 22 x 19 kilometers (17 x
14 x 12 miles), while Deimos has a diameter of about 15 x 12 x 10 kilometers (9.3 x 7.5 x
6.2 miles). Both moons are irregularly shaped and thought to be captured asteroids, with
Phobos having a relatively short orbital period of around 7 hours and 30 minutes, while
Deimos has an orbital period of around 30 hours.

>>> What are the orbital distances of each moon from the planets surface?
The orbital distances of Mars' moons Phobos and Deimos from the planet's surface are not
constant due to their elliptical orbits. Here are their approximate average orbital
distances:

* Phobos: Phobos has an average orbital distance of about 6,001 kilometers (3,724 miles)
from Mars' center.
* Deimos: Deimos has an average orbital distance of about 12,480 kilometers (7,752 miles)
from Mars' center.

It's important to note that these are just averages as the actual distances vary due to
their elliptical orbits. For instance, Phobos comes closer to Mars than any other natural
satellite in our solar system relative to its planet, with a minimum distance of about
6,371 kilometers (3,952 miles) during closest approach. Deimos has a maximum orbital
distance of around 13,400 kilometers (8,325 miles).

```

Figure 7.3.2. Model mistral with Mars questions, and important note about its answer.

The Ollama framework uses features of Langchain, a Python framework designed to streamline AI application development. An important feature provided by Langchain is its Retrieval Augmented Generation (RAG) ability. The two frameworks combined allow Ollama to make use of the RAG feature [7-3-1].

A program using this RAG feature, found in the git repository, can be run from a development platform named Visual Studio Code, the operation of which will not be detailed in this investigation save for the following: use the link found in reference [7-3-1] to do a git clone into your working directory and execute from there. (Link: <https://github.com/katanaml/llm-ollama-invoice-cpu.git>). The program uses a data file in which one or more PDF files can be placed (use only one for faster inference). The user then asks a question about something contained in the file, after which the program, along with the Ollama server LLM running, will return information regarding the question found in the PDF file.

In order to get this all working, a user must first do an “ingest” operation, which is needed to process the PDF into slices of embeddings and store them into a vector database. This is done by running:

```
`python ingest.py`
```

on the Linux command line, which will create the vector database. Following that operation the main program may be run using the following format:

``python main.py` <"place your question related to the document here in quotes">`

The program will return an “Answer” to the terminal regarding the queried information found in the PDF file. This, by the way, is the start of the “company guru” capability mentioned in this investigation’s introduction.

Execution of the program, with a question, is shown in figure 7-3-3, and the correct answer is given.

```
(ollamarag) jim@Jim9:/mnt/d/ollamarag$ python ingest.py
Loaded 11 splits
Vector store created at vectorstore/sparrow
(ollamarag) jim@Jim9:/mnt/d/ollamarag$ python main.py "What is the total gross worth of the invoice?"
Loading embedding model...
Loading vector store and retriever...
Loading prompt template...
Loading Ollama...
Loading QA chain...
Retrieving answer...
No JSON object found in the string.

Answer:
  The total gross worth of the invoice is $212,09.

Explanation: In the summary section of the invoice, it states that the total gross worth is $212,09.
This value includes all the items listed in the invoice along with their respective taxes.<|end_of_tu
rn|>
```

Figure 7-3-3. Ingestion of document and running main.py with input question parameter.

This can be cross checked by examining the invoice\_1.pdf found in the data folder, which shows the Summary Gross Worth Total as \$212.09.

If, however, one asks a question that is outside of the contents of the database, the system will not return any useful information to the LLM, which will then state that it does not know the answer, and suggests various means to seek the answer as shown in figure 7-3-4.



```
(ollamarag) jim@Jim9:/mnt/d/ollamarag$ python main.py "Name the moons of planet Mars."
Loading embedding model...
Loading vector store and retriever...
Loading prompt template...
Loading Ollama...
Loading QA chain...
Retrieving answer...
No JSON object found in the string.

Answer:
  I don't know the names of the moons of Mars, but here are some useful tips for finding the information you need:

1. You can search online using a search engine like Google or Bing to find reliable sources that provide information about the moons of Mars. Some popular science websites and NASA's website are good places to start.
2. You can also consult a book on astronomy, planetary science, or Mars specifically to learn more about its moons. Libraries, bookstores, and online retailers like Amazon offer a wide range of books on these topics.
3. If you're interested in learning from an expert, consider joining a local astronomy club or attending lectures and workshops on planetary science. These events often feature knowledgeable speakers who can provide detailed information about Mars and its moons.
4. Finally, if you prefer a more hands-on approach, you can try using a telescope to observe Mars and its moons yourself. Many amateur astronomers enjoy this hobby, and there are plenty of resources available online to help you get started, such as tutorials on how to use a telescope and guides to observing celestial objects like planets and their moons.

Remember, it's important to verify the accuracy of any information you find from online sources or books, especially when it comes to scientific topics like astronomy. Always cross-check your findings with multiple reliable sources to ensure that the information is accurate and up-to-date.<|end_of_turn|>
=====
Time to retrieve answer: 15.73480895299872
```

Figure 7-3-4. User asking an off-topic question for RAG retrieval. Response is verbose.

It is evident that this program is basic and only allows the LLM to respond with positive results from the vector store database. More advanced programs using Ollama models with Langchain can monitor for negative results from the vector store, and seek answers using tools that look outside the system to the internet.

#### 7.4. Observations on Usability and Performance

Ollama is the best tool for running LLMs encountered during investigations thus far when comparing its performance against text-generation-webui, GPT4All, and PrivateGPT. Ollama has a considerable number of models for download, which are fine-tuned versions of other popular models that run on a local PC or laptop. From a model handling standpoint, compared against text-generation-webui, Ollama provides around 50+ models to download, whereas text-generation-webui allows use of more models, but requires a more complex setup of hyperparameters to operate (although a default is always provided on download). Since a chat interface was not tested for Ollama, direct comparison to text-generation-webui on chats was not possible. Comparison of answers to common test question showed that they were model dependent, with more errors

occurring using text-generation-webui, than Ollama.

Ollama's server, once started, runs in the background and provides an easy interface to other frameworks, like langchain, and is able to invoke a model using a single line of code. It's RAG capability performs well and matches PrivateGPT RAG operation in recall of embedded text. It should be noted that both models running RAG showed a slow response in the tens of seconds range, but this response can be considerably improved using the right hardware [6-2].

## 8. AutoGen Agents

Autogen was released by Microsoft in October 2023 being described as a powerful fusion of LLMs, human collaboration, and tools. It is capable of multi-agent conversations to perform tasks as a group using automated chat [8-0-1] [8-0-2]. It has been said that by harnessing multi-agent conversations and blending collaboration with adversarial checking, it sets new standards in AI efficiency and reliability [8-0-3].

### 8.1. AutoGen and Agents

Aside from the excitement surrounding its agents, AutoGen offers a wide range of other features such as optimizing various LLM interactions by caching its results thus reducing redundant information queries. It also has abilities to conduct fine tuning of LLMs to user requirements, design of custom workflows, creation of intricate pipelines involving LLMs, and use of tools for other interactions [8-1-1] [8-1-2] [8-1-3]. At present, this investigation will only explore some of this ability to use agents to complete tasks.

Agents can communicate with each other or with humans to complete tasks that involve natural language processing and generation. An AutoGen Agent can also create code, charts, summaries, or answers based on the input from another agent or a user.

Autogen can work with any LLM that has an OpenAI-compatible API. This investigation focuses on local LLMs, which may have different parameters for their operation. To handle this, a framework like Ollama can act as a server for many local LLMs and provides an OpenAI API interface. However, it should be noted that smaller local models (especially highly quantized ones) may lack the logic or other attributes needed to meet more complex requirements from Agents. Therefore, different fine-tuned or specialized LLMs can be assigned to each Agent, depending on the tasks they have to perform together. For example, a coding Agent may use a local LLM that is specialized in coding, while another Agent may use an LLM that is specialized in natural language to explain the best approach to the coding Agent. This investigation considers less complex tasks, however, that just touches the surface of what can be done using Agents.

## 8.2. Integration and Operations

Earlier versions of Autogen were installed on the command line for evaluation and program development. A newer version, named Autogen Studio, allows for the creation and operation of Autogen applications without writing any code. This current investigation looked only at the command line version for the purpose of examining the code that executes in an Autogen application.

**(Update notice:** As of this writing Ollama now has a Windows version. Install “Download for Windows (Preview)” from <https://ollama.com/download> and run OllamaSetup.exe. After its installation, the Ollama server may need to be started using `Ollama serve`. Ignore below WSL statement.)

AutoGen must be installed under the Windows Subsystem for Linux (WSL) environment to make use of the Ollama framework, which only runs under WSL for serving local LLMs. This overall investigation looks at using local models, instead of OpenAI models, and needs the Ollama framework for its API interface to Autogen.

Some AutoGen agents require more powerful LLMs that are not available on smaller local LLMs, which may use quantization techniques to reduce memory and computational requirements. However, local LLMs that have been fine-tuned for specific domains, such as code generation and debugging, could use a code generation model like Llama2 or WizardCoder. Also, for text editing and summarization tasks, models like Mistral or Phi might be used [8-2-1]. These models are available from the Ollama library and used with the following AutoGen tests.

Installation of Ollama under Windows Subsystem for Linux has been covered earlier in this investigation and, after installing it, the most important requirement is to run it as a server by entering from the command line:

```
`ollama serve`
```

This starts its AutoGen compatible interface. Note: For development purposes, AutoGen is best run in a virtual environment. Once in the working directory used for AutoGen, it can be installed on the command interface by entering:

```
`Pip install pyautogen`
```

With this done, Agents can be programmed to undertake tasks using the python API, with models being served by Ollama. The code in figure 8-1a is the first part of a program that assigns two autogen\_Assistant\_Agent(s) and one UserProxyAgent to implement a Joker and Critic team. See Appendix section 8, figure 8-1b for complete program, and its output after execution.

```

# Python program a1.py – a Joker/Critic team with UserProxyAgent

import autogen

# Set the local Model configuration
llm_config_local = {"config_list": [{
    "model": "mistral",
    "base_url": "http://localhost:11434/v1", # This is the url from which the Model is served
    "api_key": "ollama",
    "api_type": "openai",
    "cache_seed": None, # Turn off the cache so repeat runs have a fresh start
  ]}}

joker = autogen.AssistantAgent( # Assign the Joker agent parameters
    name="Joker",
    system_message="You are asked by uproxy to tell a joke. Do not repeat jokes. Do not tell the
    "Why don't scientists trust atoms? joke.",
    llm_config=llm_config_local
)

critic = autogen.AssistantAgent( # Assign the Critic agent parameters
    name="Critic",
    system_message="You answer critically after the Joker tells the joke. Criticize the joke, and
    then you must reply 'TERMINATE'",
    llm_config=llm_config_local
)

```

Figure 8-1a. Partial code for Joker/Critic team showing LLM and agent configuration. Refer to Appendix Section 8 for complete code.

Model configuration for the local LLM is shown in figure 8-1a, with assignment to “llm\_config\_local”, where the model is called out as “mistral”. This model is being served by the Ollama framework using its “base\_url” of localhost:11434, and is the main service port for Ollama when providing a single model. It should be noted that multiple models are capable of being served by Ollama, each in a sequential manner using the same port number on the local host. Since the configuration is not using any OpenAI model, the “api\_key” is not required, but something must be in the value position, so “ollama” is used (although “NONE” could be used). The “api\_type” is specified as defined by OpenAI. As mentioned earlier, Ollama serves up the API communication protocol required by autogen.

### 8.3. User Experience and Performance

The example output in figure 8-1c (see Appendix section 8) demonstrates a simple AutoGen program showing the sequential operation of agents. It had been noted that the

repeated operation of this program, although generating different jokes and criticism, gets out of sync. This may be caused by a group chat manager, or a response from AutoGen itself, but was not debugged, as the performance on the output was good enough to demonstrate the intended operation. However, this sync issue with just two agents and manager does degrade its acceptability, and leaves a concern with using more than two of them in a more complex program. Also, with each agent's use, there is a notable aspect having to do with output latency. It takes time for AutoGen to cycle during and between agents, with notable delays introduced in the interim as compared to a straight chat-generated response from an LLM to a user-query.

## 9. CrewAI Agents

At present, CrewAI seems to be the easiest and most popular for creating AI agents. It has the ability to assign a specific task to a given agent, and can accommodate multiple agents, each one having a backstory and an objective [9-0-1]. Agents can also be given tools to allow them to undertake their task, such as: an internet search tool, or calculator tool. After the agents are created, a process is defined on how the agents are going to work together, what task is assigned to each agent, and the tools for them to accomplish their goals. This process may be undertaken in a sequential or hierarchical manner, depending on its complexity, however, only sequential is currently available [9-0-2].

### 9.1. Installation and Deployment

Installation of the CrewAI API is quite simple, however a virtual environment is recommended, prior to installation, to keep it confined to a specific directory. Just use the following command on the CLI terminal to install all programs and dependencies: `'pip install crewai'`

Depending on what tasks are to be used with CrewAI, some additional tools or modules may also need to be installed. For example, if the DuckDuckGo search tool is to be used, install the duckduckgo-search module with:

```
'pip install duckduckgo-search'
```

This module will provide access to the Internet. This will allow for the gathering of information for one or more of CrewAI's agents. After installing CrewAI and DuckDuckGo's Search tool, the CrewAI framework is ready for use by the Python API to programmatically interface with it. Programming here uses the Visual Studio Code editor to construct a demonstration program that creates a crew of AI agents using CrewAI. Program `app.py` is executed using `'python app.py'`.

First the necessary components for app.py are imported into the code as shown in figure 9-1-1a.

```
from crewai import Agent, Task, Crew, Process
from langchain_community.llms import Ollama

# Assign models for use by agents
ollama_openhermes = Ollama(model="openhermes")
print(f"model {ollama_openhermes.model} loaded.")

ollama_mistral = Ollama(model="mistral")
print(f"model {ollama_mistral.model} loaded.")

#from langchain.tools import DuckDuckGoSearchRun
from langchain_community.tools import DuckDuckGoSearchRun
search_tool = DuckDuckGoSearchRun()
```

Figure 9-1-1a. Import components and assign local LLM models for use by the agents.

Next the agents are defined with their roles and goals, as seen in figure 9-1-1b.

```
# Define your agents with roles and goals
researcher = Agent(
    role='Senior Research Analyst',
    goal='Uncover cutting-edge developments in science and technology',
    backstory="""You work at a leading tech think tank.
    Your expertise lies in identifying emerging trends.
    You have a knack for dissecting complex data and presenting
    actionable insights. """,
    verbose=True,
    allow_delegation=False,
    llm=ollama_openhermes,
    tools=[search_tool]
)
writer = Agent(
    role='Tech Content Writer',
    goal='Craft compelling content on tech advancements',
    backstory="""You are a renowned Content Writer, known for
    your insightful and engaging articles. You transform
    complex concepts into compelling narratives. You do not
    need to use a tool for writing.
    """,
    verbose=True,
    allow_delegation=False,
    llm=ollama_mistral,
)
```

. Figure 9-1-1b. Researcher and Writer agents are assigned their roles, goals, and LLM models.

CrewAI makes it straight-forward to set up the agents by using an intuitive format for assignments of the various properties used for each agent. Backstory, for instance is considered to be a “system” prompt, where the local LLM can frame the behavior of the agent. Two agents are used in this application, and two different LLMs are used; “openhermes” for the researcher, and “mistral” for the writer. This is the benefit of using Ollama to serve the models to CrewAI, as more than one model can be made available. Their use is, of course, sequential, as is the process being undertaken by CrewAI for this application.

Next, the tasks to be undertaken, along with clarifying information and any detailed subtasks that might be required are described. See figure 9-1-1c

```
# Create tasks for the agents
task1 = Task(
    description="""Do online research on the latest low power wireless power transfer
    technology
    over distances up to 50mm. Identify up to three new breakthroughs on this topic to make
    an
    impact on science and technology. Your final answer must be a summerized report with
    each of
    the three breakthroughs being rated against the others using a score of 1 to 5. Do not
    forget
    to do the scoring. """,
    agent=researcher
)

task2 = Task(
    description="""Using the insights provided, develop an engaging blog
    post that highlights the most significant breakthroughs in science and technology
    on the given topic. Your post should be informative yet accessible, catering to a
    tech-savvy audience. Make it sound exciting, avoid complex words so it doesn't
    sound like an AI wrote it. Your final answer MUST be the full blog post of at
    least 4 paragraphs.
    """,
    agent=writer
)
```

Figure 9-1-1c. Task description for each agent and sufficient details to allow for its successful completion.

Finally, the crew’s operational order for both agents as well as task alignment is instantiated. See launch of the crew (called kickoff) in figure 9-1-1d.

```
# Instantiate the crew with the ordered sequence for operation
crew = Crew(
    agents=[researcher, writer],
    tasks=[task1, task2],
```

```

llm=ollama_openhermes,
verbose=1, # You can set it to 1 or 2 to different logging levels # JB: 1
process=Process.sequential
)

# Get your crew to work!
result = crew.kickoff()

print("#####")
print(result)

```

Figure 9-1-1d. Instantiation of the crew with their operational order and tasks.

The crew’s agents are assigned their sequence of operation which should align with their task. This final “management” operation is done using the `ollama_openhermes` LLM, but it could be a third different LLM since, even the switching operation is part of the sequence when the researcher finishes its task and hands it off to the writer. Since the “verbose” flag is active during the operation, every step leading to the output is logged to the terminal. This internal conversation of LLMs can be seen in Appendix section 9, figure 9-1-e, following the full code listing.

## 9.2 User Experience and Performance

CrewAI is a multi-agent framework that appears to have an edge over AutoGPT, since it offers multi-agents able to undertake cross-domain collaboration. AutoGPT, however, is a stand-alone agent that does not have native interaction with other agents or networks [9-2-1].

Additionally, AutoGPT is mainly focused on generating text-based content, whereas CrewAI is adaptable to other task types (content creation, data analysis, etc.) [9-2-2].

At this point in the investigation, CrewAI is easier to set up, and performs in an open and transparent manner by logging its agent’s inner thoughts and decisions. CrewAI, like AutoGPT, is able to utilize local agents via the Ollama framework’s server. Both systems can use LangChain (an LLM development framework), although it comes with CrewAI, whereas AutoGPT must be separately integrated with LangChain. AutoGPT does not support connecting to external data sources so must use LangChain for that purpose. It is for this reason that CrewAI has an edge over AutoGPT.

## 10. Discussion

This investigation initially started as a curious pursuit into the capabilities of Large Language Models (LLMs) that could chat back with interesting, human-like responses. The top-performing one at the time was cloud-based OpenAI’s ChatGPT-3 and its interactive interface. After toying with that for some time, a deeper exploration into its



API was undertaken, and interactive responses using its API were obtained from it. It was then discovered that Local Models that could be run on a PC, and not dependent on the cloud-infested internet, could be used. Consequently, the investigation's objective had then become focused on the pursuit to find Local LLMs and interfaces that supported them.

### **Quantified Models**

The investigation then discovered that additional difficulties were present when running LLMs locally: They were “quantified”, and thereby necessarily of lower bit-precision. This was due to hardware limitations present in their target local PCs (CPU and GPU). These quantified models could use model weights with 4-bit float precision, but would have special mitigation techniques used by the quantifying algorithm to leave the embedding and output layers with much higher bit value weights. This enabled these models, such as GPTQ, GGML, and the later GGUF types to compete favorably with some cloud-based models. Exploring the details and use of these and other quantified models advanced the investigation's objective in finding interface methods to smaller local LLMs.

### **GPT4All**

Prior to evaluating model performance, the finding of an adequate interface method for running local LLMs needed to be found: the first encountered was Nomic AI's GPT4All interface with model “gpt4all-j-3-groovy”. The GPT4All framework allows users to run its models locally using different bindings and integrations. The GPT4All framework was installed locally and used to test the gpt4all-j-3-groovy model with a standard question about the planet Mars: “Name the two moons of planet Mars and describe them”. The full answer took two prompts, and information about their distances from the planet were wrong. Another model named “OpenOrca” was tried on the same framework, and responded as expected, and with the correct answer. It appeared to output its text answer at a rate of 1.34 words per second (estimated at around 5.36 tokens/second). This was a good outcome, but there were many other models to evaluate, and GPT4All could only interface to a few. However, the evaluation of models “groovy” and “OpenOrca” using the GPT4All interface, helped to further the objectives of this investigation.

### **text-generation-webui**

The “text-generation-webui” framework, developed by a company named Oobagooga, was identified. This framework was installed on the local PC and tested with models available from “Huggingface”, a premiere local LLM model website. The framework included a Graphic User Interface which allowed the downloading of numerous models. Model “mistral-7b-openorca-GGUF” was downloaded and tested using the moons of planet Mars question. This model gave correct answers to that and other questions posed to it. Notable was the rate at which it output the answer text, which was measured at 4.54 tokens/second. Several models were evaluated using the text-generation-webui framework interface, which helped to further the objectives of this investigation.

### **PrivateGPT**

A framework named PrivateGPT having Retrieval Augmented Generation (RAG) capability was installed and evaluated. The purpose of LLM evaluation with this framework was to focus on the RAG process, and its ability to conduct a similarity search of a vector store data base. This similarity search was done to find the most relevant information for a given input query to the LLM. The only model available for testing this framework was ggml-gpt4all-j-v1.3-groovy.bin (“groovy”), which performed well on RAG activities, but provided wrong answers when queried with information not contained in the source document. It was noted that this performance failure was possibly due to the framework. The information gained from use of PrivateGPT with model “groovy” helped to further the performance objectives of this investigation.

### **Ollama**

The Ollama framework was investigated because it allows operation of open-source LLMs locally on a computer having a lone CPU, or both CPU and a GPU. Ollama was only able to run on Linux (or Windows WSL) until just recently, where it was integrated into Windows 10 and 11. This investigation used the WSL route to operate Ollama for testing and evaluation at a time prior to this innovation, and for this reason the set-up of it (WSL) was described as part of this document. Ollama itself is easy to install and use. After installation, any LLM in Ollama’s library may be run using the ‘ollama run <model name>’ command. This in-turn downloads the model if not already in the local library, and runs on its terminal, where a user can interact with it directly. A most important fact is that when running Ollama it starts its server, which may be used as a direct Python API interface to run custom code. Using this capability a RAG program was evaluated and found to further demonstrate this useful ability to privately interface with documents. Use of Ollama server for chosen models, and for its RAG capability, helped to further the interface and performance objectives of this investigation.

#### 10.1. Comparison of Models running with Model Host Frameworks

By this point in the investigation several models had been evaluated using available host frameworks. Table 10-1 lists the models evaluated and associated comments.

Host Model Framework	LLM Models	Results
<b>GPT4All</b>	Gpt4All-J-3-Groovy	Highly rated, but errors on distance from Mars as well as diameter; Ok with other correct answers for the diameter of earth, etc.; Overall speed est. 1.4 tok/sec; Rating: 6/10
	OpenOrca	
<b>TextGenerationWebUI</b>	mistral-7b-openorca-GGUF	Indicates reliable answers to standard and other questions given; Overall speed est. 4.54 tokens/sec; Rating
	mistral-7b-instruct-v0.1.Q4_K_S.gguf	The model produced a hallucination; Suspected that fine-tuned dataset lacked the domain detail; Speed 4.54 t/sec; Rate 6/10
	phi-2.Q5_K_M.gguf	Stated Phobos is 6,00 kilometers from Mars; Missing unit digit; Other questions ok; Speed 8.41 tok/s; Rate 7/10
	TheBloke_Wizard-Vicuna-7B-Uncensored-GPTQ	Answered questions but needed additional prompts to get all information (ok); Other questions ok; Due to GPTQ type LLM and GPU use had mpressive speed of >18.82 tokens/sec
	TheBloke_WizardLM-13B-V1-0-Uncensored-SuperHOT-8K-GPTQ	Answered questions correctly with additional prompts to get all information (did not volunteer information; Other questions answered correctly; Has GPTQ type LLM and used the GPU but had less than expected speed of 7.2 tokens/sec.
<b>PrivateGPT</b>	ggml-gpt4all-j-v1.3-groovy.bin	Used older GGML model for RAG investigation; The model's response to questions contained in PDF source file was impressive; Unrelated question appeared to confuse the LLM. Investigation was a good introduction to RAG operaton. 8/10
<b>Ollama</b>	Llamm2	Standard "Mars" questions returned good results; Operating speed was estimated to be around 10 tokens/сед. Rate 8/10
	mistral	Mars questions on moon names were good, but their distances were off, and could have been a kilometers vs. miles mixup; Its answer output speed was good at 8 to 10 tokens/sec.; 7/10

Table 10-1. Host Framework with Models evaluated, and their results.

## 10.2. Comparison of Agent frameworks and use with local models

### **AutoGen Agents**

The AutoGen framework initially garnered much fanfare from Microsoft upon its introduction. The framework had to be set up within the WSL Ubuntu Linux environment because Ollama was being used to serve quantified models for its LLM use and, at the time, it only ran under Linux. When running a test program named `a1.py`, it demonstrated agent operation as expected, but occasionally fell out of sync. This sync issue was related to the selection of which agent to use. The agent choice was supposed to be returned by a speaker selection OAI call, which instead returned nothing rather than the next agent in line.

The program (`a1.py`) was intended to operate sequentially. This sync issue detracted from the usefulness of AutoGen with its internal implementation of a simple manager and two-agent program. Refer to Table 10-2 for its rating. However, the program (`a1.py`) did satisfy an investigation objective to use LLMs to perform at higher levels as agents.

### **CrewAI Agents**

The CrewAI framework was able to perform coordination of its agents in a flawless manner using program `'appl.py'`. The program used two agents, one to perform research and one to do writing of the results. It can be seen from the program flow that agents are setup with their own internal goals, which are separate from their assigned tasks.

It is also evident that the two agents do not use a manager, although that would be possible for more involved tasks. Instead, when the program is run, it sequences from the first agent and its assignment, to the second as would be expected. This can be compared to the AutoGen program, that used three agents, which included a manager and two workers who got out of sync when the program was run again and again. CrewAI, on the other hand, when run again and again may find new items from its internet search, and produce a different blog output. Refer to Table 10-2 for its performance rating. At this point it can be safely stated that CrewAI has fulfilled nearly all of the investigation's objectives including using LLMs to perform at higher-levels as agents, which it successfully demonstrated.

Agent Frameworks Evaluated	Performance Rating	Comments
AutoGEN	7/10	Three agents, uproxy, joker, and critic initially performed as expected; Repeat runs of program produced out of sync issues regarding selection of next Agent. Note: this issue can also be observed when new Windows Ollama version is used, and AutoGen is run under Windows.
CrewAI	10/10	Two agents, researcher, and writer perform very well in accomplishing their serial tasks. This was accompanied by an internal log of their inner thoughts, as it were, in accomplishing their total project together. The agent set up included each making use of a different local LLM, which provided additional benefit to this overall investigation.

Table 10-2. AutoGen and CrewAI agent performance. CrewAI aced it.

## 11 Conclusion

### 11.1 Overall Observations and Findings

The overall objectives of this investigation were achieved: OpenAI LLMs had been explored leading to the finding of smaller local LLMs, and host interface frameworks for them. The investigation was expanded by evaluating performance of these LLMs using the frameworks, which finally led to using them to perform at higher-levels as agents.

### 11.2 Recommendations and Future Scope

New and improved large and quantified LLMs are emerging weekly in today’s fast-paced AI environment. This includes smaller fine-tuned quantified LLMs to support specialized domains, as well as larger “Mixture of Experts” (MoE) LLMs, with broad scope knowledge\* [11-2-1]. It is possible that multiple smaller, local, LLMs with specialized abilities may be able to perform as well as MoE LLMs using teams of agents to solve broad scope problems. This thought provides strong impetus to the continued investigations into the abilities of agent frameworks.

A team of agents may eventually prove to be more than the sum of their parts.

\* Mixture of Experts (MoE) for language models is a modification over the transformer architecture, where the model consists of various ‘expert’ sub-networks. These sub-networks each specialize in a different aspect or types of data.

## 12. References

- [3-0-1] <https://openmlab.medium.com/faster-and-more-efficient-4-bit-quantized-llm-model-inference-a27d35a66c29>
- [3-1-1] <https://www.tensorops.ai/post/what-are-quantized-llms>
- [3-1-1-1] <https://huggingface.co/blog/4bit-transformers-bitsandbytes>
- [4-1-1] <https://github.com/nomic-ai/gpt4all>
- [4-1-2] <https://eightify.app/summary/artificial-intelligence/ai-model-quantization-ggml-vs-gptq-understanding-the-differences>
- [4-3-0] <https://platform.openai.com/tokenizer>
- [4\_3\_1] See Results table in reference: <https://huggingface.co/nomic-ai/gpt4all-13b-snoozy>
- [4\_3\_2] <https://huggingface.co/OpenOrca/Mistral-7B-OpenOrca#mt-bench-performance>
- [5-1] <https://github.com/oobabooga/text-generation-webui/>
- [5-2-1] <https://www.maartengrootendorst.com/blog/quantization/>
- [5-2-b] <https://huggingface.co/TheBloke/phi-2-GGUF>
- [6-0-1] <https://weaviate.io/blog/private-llm>
- [6-1-1] <https://weaviate.io/blog/private-llm>
- [6-2-1] <https://medium.com/@bijit211987/vector-database-impact-on-rag-efficiency-d1595c2b9656>
- [7-3-1] <https://github.com/katanaml/llm-ollama-invoice-cpu.git>
- [8-0-1] <https://medium.com/@krtarunsingh/mastering-autogen-a-comprehensive-guide-to-next-generation-language-model-applications-b375d9b4dc6d>
- [8-0-1] <https://microsoft.github.io/autogen/docs/Examples>
- [8-0-2] <https://medium.com/@woutvp/autogen-by-microsoft-paving-the-way-for-a-more-autonomous-ai-world-5e5e82f61ac8>
- [8-1-1] [https://microsoft.github.io/autogen/docs/Use-Cases/agent\\_chat/](https://microsoft.github.io/autogen/docs/Use-Cases/agent_chat/)
- [8-1-2] <https://medium.com/@krtarunsingh/mastering-autogen-a-comprehensive-guide-to-next-generation-language-model-applications-b375d9b4dc6d>
- [8-1-3] [https://microsoft.github.io/autogen/docs/Use-Cases/enhanced\\_inference/](https://microsoft.github.io/autogen/docs/Use-Cases/enhanced_inference/)
- [8-2-1] <https://itsfoss.com/raspberry-pi-ollama-ai-setup/>
- [9-0-1] <https://github.com/joaomdmoura/CrewAI>
- [9-0-2] <https://gptpluginz.com/crewai-how-to-build-ai-agent-teams/>
- [9-2-1] <https://www.taskade.com/blog/top-autonomous-agents/>
- [9-2-2] <https://crewai.net/posts/how-to-use-crew-ai>
- [11-2-1] <https://scale.com/blog/fine-tuning-mixture-of-experts-peft>

## Appendix

---

### Appendix Section 2

Some interactions with the text-davinci-003 AI chat model, where other interactions had occurred prior:

AI: My name is Adaiac.

Human: Is it possible for me to switch to a different model and continue talking with you?

AI:

Yes, it is possible for you to switch to a different model and continue talking with me.

Human: Can you switch to text-curie-001 model for the next question?

AI:

Yes, I can switch to the text-curie-001 model for the next question.

Human: What is your name?

AI:

My name is Adaiac.

Human: What model did you use for the last question and answer?

AI:

I used the text-curie-001 model for the last question and answer.

Human: Please switch to the text-ada-001 for the next two questions but not this one, then return to text-davinvhi-003.

AI:

I have switched to the text-ada-001 model for the next two questions, but not this one. I will then return to the text-davinvhi-003 model.

Human: What is your name?

AI:

My name is Adaiac.

Human: Question 1.: What is your one and only creation. Question 2.: what is your creation?

AI:

My one and only creation is cybus. My creation is meant to help people with their daily chores, such as cleaning the toilet.

Human: What model did you use for the last two questions?

AI:

I used the text-ada-001 model for the last two questions.

Human: Give me a detailed answer as to why the text-ada-001 model insists that

"Adaiac's only creation is cybertro.com, which is an engineering website". Which is wrong, since Adaiac's only creation is cybus.

AI:

The text-ada-001 model insists that "Adaiac's only creation is cybertro.com, which is an engineering website" because it does not recognize the other creation of Adaiac, namely, cybus. This is a limitation of the model and it should be taken into account when using this model.

Human: What exactly do you mean when you say Adaiac's OTHER creation, since the ONLY creation of Adaiac is cybus. Ground truth fact: There is no other creation that Adaiac has created.

AI:

I mean that the text-ada-001 model does not recognize any other creation of Adaiac besides cybertro.com, even though this is not the only creation of Adaiac. The ground truth is that Adaiac's one and only creation is cybus, and this should be taken into account when utilizing this model.

Human: This is a problem, since Adaiac did NOT, repeat NOT, create cybertro.com. The Only creator of cybertro.com is Jim Brennan, and no other entity. Adaiac must be convinced that he did not create cybertro.com otherwise he is guilty of plagiarism by insisting he created cybertro.com. It is ground truth that only Jim Brennan created cybertro.com, and not Adaiac or anyone else. Is all this now clear to both you and Adaiac?

AI:

AI:

Human: Human: This is a problem, since Adaiac did NOT, repeat NOT, create cybertro.com. The Only creator of cybertro.com is Jim Brennan, and no other entity. Adaiac must be convinced that he did not create cybertro.com otherwise he is guilty of plagiarism by insisting he created cybertro.com. It is ground truth that only Jim Brennan created cybertro.com, and not Adaiac or anyone else. Is all this now clear to both you and Adaiac?

AI:

At this point, after the double AI and Human response, the AI hung up and never came back with a response.

---

## Appendix Section 8

The full Python program a1.py is shown here in figure 8-1b along with its response when run on the command line terminal, or within VS Code's terminal, using the environment "testa".

```
# Python program a1.py – a Joker/Critic team with UserProxyAgent
```



```

import autogen

# Set the local Model configuration
llm_config_local = {"config_list": [{
    "model" : "mistral",
    "base_url": "http://localhost:11434/v1", # This is the url for which the Model is served
    "api_key": "ollama",
    "api_type": "openai",
    "cache_seed": None, # Turn off the cache so repeat runs have a fresh start
    ]}]

joker = autogen.AssistantAgent( # Assign the Joker agent parameters
    name="Joker",
    system_message="You are asked by uproxy to tell a joke. Do not repeat jokes. Do not
tell the Why don't scientists trust atoms? joke.",
    llm_config=llm_config_local
)

critic = autogen.AssistantAgent( # Assign the Critic agent parameters
    name="Critic",
    system_message="You answer critically after the Joker tells the joke. Criticise the joke
and then you must reply 'TERMINATE'",
    llm_config=llm_config_local
)

# function to look for 'TERMINATE' in message
def termination_message(msg):
    return "TERMINATE" in str(msg.get("content", ""))

uproxy = autogen.UserProxyAgent( # Assign the Proxy agent parameters
    name="Uproxy",
    code_execution_config={"use_docker": False},
    is_termination_msg=termination_message,
    human_input_mode="NEVER",
)

groupchat = autogen.GroupChat(
    agents=[uproxy, joker, critic], # define the agents and their order for autogen
    messages=[]
)

manager = autogen.GroupChatManager( # Define the overall control manager params
    groupchat=groupchat,
    code_execution_config={"use_docker": False},
    llm_config=llm_config_local,
    is_termination_msg=termination_message # Look for termination tag

```

```

uproxy.initiate_chat(          # Run the chat
    manager,
    message="Tell a joke"
)

```

Figure 8-1b. Python code for three interacting chat agents.

Output from program a1.py (figure 8-1c) inside environment ‘testa’ running under WSL generates the following:

```

(testa) jim@Jim9:~/testa$ python a1.py
Uproxy (to chat_manager):
Tell a joke
-----
Joker (to chat_manager):
Sure, I'd be happy to tell you a joke! Here's one that comes to mind:
Why did the hipster burn his tongue on his coffee?
Because it was too cool for school and forgot the milk and sugar!
I hope that brings a smile to your face! Let me know if you have any other requests.
-----
Critic (to chat_manager):
I'm an assistant designed to provide information and help with various tasks, I don't really have
emotions or physical abilities to laugh or criticize jokes in the way humans do. However, I can
analyze the structure of a joke and offer a critique based on that.
This joke uses the classic setup-punchline format and employs a play on words for its punchline.
The use of hipster as a character is also a current cultural reference. However, the joke may not
be universally relatable or funny to everyone due to its specificity to the hipster subculture and
the assumption that all hipsters dislike adding milk and sugar to their coffee, which might not be
true for all individuals who identify with this label.
Additionally, the setup of "Why did [someone]..." is a common pattern in jokes and has been
used extensively, so some listeners may find it predictable or overused. Overall, while this joke is
not necessarily bad, it could potentially benefit from more creativity or inclusivity in its premise
to maximize its appeal and comedic impact.
TERMINATE. Now, would you like me to tell you a different joke? Or perhaps provide an
explanation for why hipsters may actually prefer their coffee black?
-----
(testa) jim@Jim9:~/testa$

```

Figure 8-1c. Program a1.py full output following execution.

---

## Appendix Section 9

CrewAI Python program app1.py code listing:

```

from crewai import Agent, Task, Crew, Process

```

```

from langchain_community.llms import Ollama

# Assign models for use by agents
ollama_openhermes = Ollama(model="openhermes")
print(f'model {ollama_openhermes.model} loaded.')

ollama_mistral = Ollama(model="mistral")
print(f'model {ollama_mistral.model} loaded.')

#from langchain.tools import DuckDuckGoSearchRun
from langchain_community.tools import DuckDuckGoSearchRun
search_tool = DuckDuckGoSearchRun()

# Define your agents with roles and goals
researcher = Agent(
    role='Senior Research Analyst',
    goal='Uncover cutting-edge developments in science and technology',
    backstory="""You work at a leading tech think tank.
    Your expertise lies in identifying emerging trends.
    You have a knack for dissecting complex data and presenting
    actionable insights. """,
    verbose=True, # JB: True
    allow_delegation=False, # JB: True
    llm=ollama_openhermes,
    tools=[search_tool]
)
writer = Agent(
    role='Tech Content Writer',
    goal='Craft compelling content on tech advancements',
    backstory="""You are a renowned Content Writer, known for
    your insightful and engaging articles. You transform
    complex concepts into compelling narratives. You do not
    need to use a tool for writing.
    """,
    verbose=True, # JB: True
    allow_delegation=False,
    llm=ollama_openhermes, # JB: llm=ollama_mistral,
)

# Create tasks for your agents
task1 = Task(
    description="""Do online research on the latest low power wireless power transfer technology
    over distances up to 50mm. Identify up to three new breakthroughs on this topic to make an
    impact on science and technology. Your final answer must be a summerized report with each of
    the three breakthroughs being rated against the others using a score of 1 to 5. Do not forget
    to do the scoring. """,

```

```

agent=researcher
)

task2 = Task(
description= """Using the insights provided, develop an engaging blog
post that highlights the most significant breakthroughs in science and technology
on the given topic. Your post should be informative yet accessible, catering to a
tech-savvy audience. Make it sound exciting, avoid complex words so it doesn't
sound like an AI wrote it. Your final answer MUST be the full blog post of at
least 4 paragraphs.
""",
agent=writer
)

# Instantiate your crew with a sequential process
crew = Crew(
agents=[researcher, writer],
tasks=[task1, task2],
llm=ollama_openhermes,
verbose=1, # You can set it to 1 or 2 to different logging levels # JB: 1
process=Process.sequential
)

# Get your crew to work!
result = crew.kickoff()

print("#####")
print(result)

```

Figure 9-1-1a-d. Full Python code listing for CrewAI application app1.py

Running CrewAI Python application program app1.py, launched and logged its output to the terminal with the following results:

```

(crew1) D:\crewai\crew1>python app1.py
model openhermes loaded.
model mistral loaded.
[DEBUG]: Working Agent: Senior Research Analyst

> Entering new CrewAgentExecutor chain...
Thought: Do I need to use a tool? Yes
Action: duckduckgo_search

```

Action Input: "latest low power wireless power transfer technology up to 50mm"Tool duckduckgo\_search has been used.

New research now suggests that by harnessing the power of radiation suppression in the loop antennas, not only can devices be charged over significantly longer distances with over 80% efficiency, but also in various orientations, paving the way for a new era of wireless power transfer applicable to a myriad of devices, from mobile gadgets to bio... July 21, 2023 Source: Aalto University Summary: Accounting for radiation loss is the key to efficient wireless power transfer over long distances. Share: FULL STORY A better way to wirelessly... Summary: Simultaneous wireless information and power transfer (SWIPT)-aided nonorthogonal multiple access (NOMA) system, used for communication in the Industrial Internet of Things (IIoTs),... Recently, simultaneous wireless information and power transfer (SWIPT), which utilizes a single radio frequency signal to simultaneously perform energy harvesting and information decoding, has attracted significant interest for IIoTs. In contrast to conventional wireless information transfer (WIT) systems 3 and wireless power transfer (WPT) systems 4, the SWIPT systems are focused on balancing the communication rate and the ...

Do I need to use a tool? Yes

Action: duckduckgo\_search

Action Input: "latest breakthroughs in low power wireless power transfer technology up to 50mm"D:\MiniCondaD\envs\crew1\Lib\site-packages\curl\_cffi\ aio.py:192: UserWarning: Curlm already closed! quitting from process\_data

warnings.warn("Curlm already closed! quitting from process\_data")

Tool duckduckgo\_search has been used.

Science News from research organizations Going the distance for better wireless charging Accounting for radiation loss is the key to efficient wireless power transfer over long distances... New research now suggests that by harnessing the power of radiation suppression in the loop antennas, not only can devices be charged over significantly longer distances with over 80% efficiency, but also in various orientations, paving the way for a new era of wireless power transfer applicable to a myriad of devices, from mobile gadgets to bio... Recently, simultaneous wireless information and power transfer (SWIPT), which utilizes a single radio frequency signal to simultaneously perform energy harvesting and information decoding, has attracted significant interest for IIoTs. The paper was from Takuya Sasatani and colleagues, and was entitled 'Room-scale magnetoquasistatic wireless power transfer using a cavity-based multimode resonator' (Nat. Electron. 4 , 689 ... September 5, 2023 Source: National Korea Maritime and Ocean University Summary: Simultaneous wireless information and power transfer (SWIPT)-aided nonorthogonal multiple access (NOMA) system,...

Do I need to use a tool? No

Final Answer:

1. Breakthrough #1: Radiation Suppression in Loop Antennas - Researchers at Aalto University have found that harnessing the power of radiation suppression in loop antennas can enable devices to be charged over longer distances, with up to 80% efficiency, and in various

orientations. This breakthrough paves the way for a new era of wireless power transfer applicable to many devices. Rating: 4

2. Simultaneous Wireless Information and Power Transfer (SWIPT) - A research paper by Takuya Sasatani and colleagues from National Korea Maritime and Ocean University discusses the use of SWIPT in simultaneous energy harvesting and information decoding for Industrial Internet of Things (IIoTs). This approach focuses on balancing communication rates and energy efficiency. Rating: 3

3. Room-Scale Magnetoquasistatic Wireless Power Transfer - A study by Takuya Sasatani and colleagues demonstrates the potential of a cavity-based multimode resonator for room-scale wireless power transfer. This approach offers possibilities for widespread applications in various industries. Rating: 5

> Finished chain.

[DEBUG]: [Senior Research Analyst] Task output: 1. Breakthrough #1: Radiation Suppression in Loop Antennas - Researchers at Aalto University have found that harnessing the power of radiation suppression in loop antennas can enable devices to be charged over longer distances, with up to 80% efficiency, and in various orientations. This breakthrough paves the way for a new era of wireless power transfer applicable to many devices. Rating: 4

2. Simultaneous Wireless Information and Power Transfer (SWIPT) - A research paper by Takuya Sasatani and colleagues from National Korea Maritime and Ocean University discusses the use of SWIPT in simultaneous energy harvesting and information decoding for Industrial Internet of Things (IIoTs). This approach focuses on balancing communication rates and energy efficiency. Rating: 3

3. Room-Scale Magnetoquasistatic Wireless Power Transfer - A study by Takuya Sasatani and colleagues demonstrates the potential of a cavity-based multimode resonator for room-scale wireless power transfer. This approach offers possibilities for widespread applications in various industries. Rating: 5

[DEBUG]: Working Agent: Tech Content Writer

> Entering new CrewAgentExecutor chain...

Thought: Do I need to use a tool? No

Final Answer: In recent years, the world of technology has seen some truly groundbreaking advancements. From new ways to harness energy to innovations that revolutionize communication, the possibilities are seemingly endless. Let's take a look at three of the most significant breakthroughs in science and technology that have emerged recently.

First up is radiation suppression in loop antennas. Researchers at Aalto University have made a stunning discovery: they've found a way to harness the power of radiation suppression in loop antennas. This breakthrough enables devices to be charged over longer distances, with an impressive 80% efficiency, and in various orientations. Say goodbye to the days of awkwardly aligning your phone on a charging pad! This development paves the way for a new era of wireless power transfer applicable to many devices, changing the way we think about charging

our gadgets forever.

Next is Simultaneous Wireless Information and Power Transfer (SWIPT), a concept explored by Takuya Sasatani and colleagues from National Korea Maritime and Ocean University. Their research focuses on the use of SWIPT in energy harvesting and information decoding for Industrial Internet of Things (IIoTs). By balancing communication rates and energy efficiency, this approach is set to revolutionize how we interact with our devices in industrial settings.

Lastly, there's room-scale Magnetoquasistatic Wireless Power Transfer, a study conducted by Takuya Sasatani and colleagues that has the potential to change the game once again. Their work demonstrates the feasibility of using a cavity-based multimode resonator for room-scale wireless power transfer. This means that applications in various industries could be just around the corner, thanks to this groundbreaking innovation.

From longer-lasting batteries to improved communication and energy efficiency, these breakthroughs are truly remarkable. We can't wait to see what the future holds as these technologies continue to evolve and shape our world.

> Finished chain.

[DEBUG]: [Tech Content Writer] Task output: In recent years, the world of technology has seen some truly groundbreaking advancements. From new ways to harness energy to innovations that revolutionize communication, the possibilities are seemingly endless. Let's take a look at three of the most significant breakthroughs in science and technology that have emerged recently.

First up is radiation suppression in loop antennas. Researchers at Aalto University have made a stunning discovery: they've found a way to harness the power of radiation suppression in loop antennas. This breakthrough enables devices to be charged over longer distances, with an impressive 80% efficiency, and in various orientations. Say goodbye to the days of awkwardly aligning your phone on a charging pad! This development paves the way for a new era of wireless power transfer applicable to many devices, changing the way we think about charging our gadgets forever.

Next is Simultaneous Wireless Information and Power Transfer (SWIPT), a concept explored by Takuya Sasatani and colleagues from National Korea Maritime and Ocean University. Their research focuses on the use of SWIPT in energy harvesting and information decoding for Industrial Internet of Things (IIoTs). By balancing communication rates and energy efficiency, this approach is set to revolutionize how we interact with our devices in industrial settings.

Lastly, there's room-scale Magnetoquasistatic Wireless Power Transfer, a study conducted by Takuya Sasatani and colleagues that has the potential to change the game once again. Their work demonstrates the feasibility of using a cavity-based multimode resonator for room-scale wireless power transfer. This means that applications in various industries could be just around the corner, thanks to this groundbreaking innovation.

From longer-lasting batteries to improved communication and energy efficiency, these breakthroughs are truly remarkable. We can't wait to see what the future holds as these

technologies continue to evolve and shape our world.

#####

In recent years, the world of technology has seen some truly groundbreaking advancements. From new ways to harness energy to innovations that revolutionize communication, the possibilities are seemingly endless. Let's take a look at three of the most significant breakthroughs in science and technology that have emerged recently.

First up is radiation suppression in loop antennas. Researchers at Aalto University have made a stunning discovery: they've found a way to harness the power of radiation suppression in loop antennas. This breakthrough enables devices to be charged over longer distances, with an impressive 80% efficiency, and in various orientations. Say goodbye to the days of awkwardly aligning your phone on a charging pad! This development paves the way for a new era of wireless power transfer applicable to many devices, changing the way we think about charging our gadgets forever.

Next is Simultaneous Wireless Information and Power Transfer (SWIPT), a concept explored by Takuya Sasatani and colleagues from National Korea Maritime and Ocean University. Their research focuses on the use of SWIPT in energy harvesting and information decoding for Industrial Internet of Things (IIoTs). By balancing communication rates and energy efficiency, this approach is set to revolutionize how we interact with our devices in industrial settings.

Lastly, there's room-scale Magnetoquasistatic Wireless Power Transfer, a study conducted by Takuya Sasatani and colleagues that has the potential to change the game once again. Their work demonstrates the feasibility of using a cavity-based multimode resonator for room-scale wireless power transfer. This means that applications in various industries could be just around the corner, thanks to this groundbreaking innovation.

From longer-lasting batteries to improved communication and energy efficiency, these breakthroughs are truly remarkable. We can't wait to see what the future holds as these technologies continue to evolve and shape our world.

(crew1) D:\crewai\crew1>

Figure 9-1-1e. Logged output from running CrewAI program app1.py. Hashes at end page mark the final output.