Run AI Software Locally

Do you want to run AI models locally? You can. No specialised hardware is required, although you need a reasonably powerful "desktop" computer. Nevertheless, you can try it on low-end consumer machines. You can run some AI software suits on a low-power computer like mine; the only problem with this approach is that the software will sorely test your patience. You need a computer that has at least 16GB DDR3 RAM or better, a quad-core (four cores + four threads) CPU capable of running at 2.1GHz, 8.5GB storage space for models, and 200MB for the software. No fancy GPU is required. I tested it on my Windows desktop computer. Everything went okay.

My PC's specs:

```
Processor:      Intel(R) Core(TM) i5-4570S CPU @ 2.90GHz   2.90 GHz
                Haswell 22nm Technology
Installed RAM:  16.0 GB (15.5 GB usable) Dual-Channel
                DDR3 @ 781MHz
Board:          Gigabyte Technology Co. Ltd. H81M-S
Graphics:       ACER S271HL (1366x768@60Hz)
                Intel HD Graphics 4600 (Gigabyte)
                512MB NVIDIA GeForce 9600 GT (XFX Pine Group)
                ForceWare version: 342.01
                SLI Disabled
System type:    64-bit operating system, x64-based processor
Pen and touch:  No pen or touch input is available for this display
```

OS:

```
Edition:         Windows 10 Pro
Version:         20H2
Installed on:    9/12/2020
OS build:        19042.1055
Experience:      Windows Feature Experience Pack 120.2212.2020.0
```

GPT4All

Luckily, GPT4All offers you the "easy" solution.

Check whether your PC's CPU supports AVX2:

GPT4All is a GUI application. The models must be downloaded separately. No worry! You will find links to those models on their website. The models GPT4All supports are also supported by another program LlamaGPTJ-chat. The program LlamaGPTJ-chat is a command-line program. It is a model runner that also checks whether your CPU supports AVX2. On top of that, you will need this small C++ program later if you wish to use the models via this program from your console. Many people love console applications, too. So, there's no loss.

Step 1:

Download LlamaGPTJ-chat's compiled executable files (depending on your OS) from the 'Release' section. For example: if you are on Windows, you'll need 1. chat-windows-latest-avx.exe and 2. chat-windows-latest-avx2.exe.

Open a terminal emulator of your choice, cmd.exe, WezTerm, Alacritty etc., in the directory where you've downloaded the executable files.

Type `chat-windows-latest-avx2.exe`.

```
C:\Users\USERNAME>chat-windows-latest-avx2.exe
LlamaGPTJ-chat (v. 0.2.1)
Your computer supports AVX2
LlamaGPTJ-chat: loading .\models\ggml-vicuna-13b-1.1-q4_2.bin
.
```

If you see `Your computer supports AVX2`, that means you are good to go with the regular versions of the model runner applications (both the GPT4All and LlamaGPTJ-chat).

If not, Download the "AVX Only" version of GPT4All and keep the "chat-windows-latest-avx.exe" version of LlamaGPTJ-chat, since your CPU has no support for AVX2.

Step 2:

Download models:

The basic models:

1. ggml-mpt-7b-chat.bin
2. ggml-gpt4all-j-v1.3-groovy.bin

There is no shortage of pre-trained models on their site for your use. These are the models GPT4All will first search for. Download the models with a download manager like Free Download Manager. Usually, downloading such large files (4.52GB and 3.92GB, respectively) without the torrent protocol requires a download manager. FDM is a closed-source, proprietary, free-to-use file downloader application.

Step 3:

Install GPT4All. It's fairly easy.

Copy chat-windows-latest-avx.exe or chat-windows-latest-avx2.exe (depending on the situation) into the `bin` directory of the installation folder of GPT4All, generally, `%USERPROFILE%\gpt4all\bin`.

Step 4:

Add the `%USERPROFILE%\gpt4all\bin` folder to PATH:

`WINDOWS` + `r` -> type `systempropertiesadvanced` -> `ENTER`.

`Advanced` Tab -> `Environment Variables...` -> `System variables...` -> `Path` -> `New`. Paste the actual path there, `C:\Users\YOUR_USERNAME\gpt4all\bin`.

Step 5:

Create a hard symbolic link to a folder in another location of your hard drive where you will store the large models. I wanted to keep the downloaded model files (`*.bin`) in `N:\GPT4All`. The program expects the model files to be inside `C:\Users\YOUR_USERNAME\AppData\Local\nomic.ai\GPT4All\`. Sorry! I don't want those large files to occupy the system drive of my machine. I created a symbolic link. `mklink` is used to create symlinks in Windows.

```
mklink /Option Link Target
Or,
mklink /Option Virtual Actual

Options:
/D -> soft link
/H -> hard link
/J -> hard link pointing to a directory (directory junction)
```

Here, in my case, it was:

```
mklink /J "C:\Users\YOUR_USERNAME\AppData\Local\nomic.ai\GPT4All\" "N:\GPT4All"
```

Start `cmd.exe` in Administrator mode.

`WINDOWS` + `r` -> type `cmd` -> `CTRL + SHIFT + ENTER`.

Issue the appropriate command depending on where you want to store the downloaded models.

Now you can drop all model files in that folder on another partition. Do that now.

Step 6:

Create a `json` file for LlamaGPTJ-chat to avoid typing lots of commands before using LlamaGPTJ-chat.

The file should be created in `C:\Users\YOUR_USERNAME\gpt4all\`.

```
type nul >> "%USERPROFILE%\gpt4all\chat.json"
```

```
notepad "%USERPROFILE%\gpt4all\chat.json"
```

Paste the contents given below.

```
{
// CMD in Admin mode
// mklink /J "C:\Users\AppuRaja\AppData\Local\nomic.ai\GPT4All\" "N:\GPT4All"
// https://gpt4all.io/
// https://github.com/kuvaus/LlamaGPTJ-chat
// chat-windows-latest-avx2.exe -j %userprofile%\gpt4all\chat.json

//"model": "N:\GPT4All\ggml-gpt4all-j-v1.3-groovy.bin",
//"model": "N:\GPT4All\ggml-mpt-7b-chat.bin",
//"model": "N:\GPT4All\ggml-nous-gpt4-vicuna-13b.bin",
//"model": "N:\GPT4All\ggml-vicuna-7b-1.1-q4_2.bin",
//"model": "N:\GPT4All\ggml-wizardLM-7B.q4_2.bin",
//"model": "N:\GPT4All\ggml-nous-gpt4-vicuna-13b.bin",

"load_json": "N:\GPT4All\gpt4all\chat.json"
"load_log": "N:\GPT4All\chatlog.log",
"save_log": "N:\GPT4All\chatlog.log",
"n_predict": 100,
"top_p": 1.0,
"top_k": 50400,
"temp": 0.9,
"batch_size": 0.9,
"repeat_penalty": 1.1,
"n_batch": 20,
"threads": 4,
"model": "N:\GPT4All\ggml-mpt-7b-chat.bin",
"no-interactive": "false"
}
```

Step 7:

Run GPT4All. Find it in your Start Menu.

To run LlamaGPTJ-chat command-line model runner, issue this command:

```
chat-windows-latest-avx2.exe -j %USERPROFILE%\gpt4all\chat.json
```

Or, simply:

```
chat-windows-latest-avx2.exe -m "N:\GPT4All\ggml-mpt-7b-chat.bin" -t 4
```

A WinBATCH/BASH/FISH/ZSH script should work as well. The AI model `ggml-wizardLM-7B.q4_2.bin` generated almost perfect code, the last time I checked.

Feel free to experiment with the parameters. Change models and parameters in the `json` file. You need the `json` file to quickly insert flags into the LlamaGPTJ-chat model runner. The `json` file is unrelated to the GPT4All GUI model runner program.

You might want to keep an eye on a similar program koboldcpp. It can run many AI model formats. The program is small, nearly 20MB.

For more details on koboldcpp:

https://github.com/LostRuins/koboldcpp

https://huggingface.co/TheBloke/WizardCoder-15B-1.0-GGML/blob/main/README.md

https://huggingface.co/TheBloke/WizardCoder-15B-1.0-GPTQ

https://huggingface.co/TheBloke/MPT-7B-Storywriter-GGML/discussions/2#6475d914e9b57ce0caa68888

Running koboldcpp:

```
koboldcpp.exe --help
```

```
koboldcpp.exe --model "N:\GPT4All\ggml-wizard-13b-uncensored.bin" --smartcontext --nommap
```

```
koboldcpp.exe --model "N:\GPT4All\WizardCoder-15B-1.0.ggmlv3.q4_1.bin" --smartcontext --nommap
```

Drop koboldcpp's compiled binary in the same folder where you have your LlamaGPTJ-chat (`chat-windows-latest-avx2.exe`).

Run small LLMs on low-end hardware under Linux (even without AVX/AVX2 support and GPUs)

Most of the instructions you'll see in this part are inspired by a GitHub page demonstrated by Gary Explains. This gentleman has tested the process of running LLMs for us on low-end hardware devices such as a Raspberry Pi 4 8GB. It works on ARM CPUs, as well as x86 CPUs. Visit his page to read the process in detail. Here's a quick overview of the steps:

How to Run a ChatGPT-like AI Bot on a Raspberry Pi

Models will be downloaded from:

1. Hugging Face - TheBloke/Llama-2-7b-Chat-GGUF: https://huggingface.co/TheBloke/Llama-2-7b-Chat-GGUF/tree/main
2. Hugging Face - TheBloke/CodeLlama-7B-Instruct-GGUF: https://huggingface.co/TheBloke/CodeLlama-7B-Instruct-GGUF/tree/main

Thanks to Hugging Face for providing the LLMs.

The model runner application in this instance is llama.cpp. It has to be built before it can be used. Make sure you have Python 3 on the system and you can access Python from the terminal: `python3 --version`. On my Debian machine, the version of Python found is 3.11.2.

Set up your build environment:

Update the system and the installed packages.

```
sudo apt update
```

Install the C/C++ compiler toolchain.

```
sudo apt install bash git g++ wget build-essential sakura
```

Clone the source code of the model runner application.

```
git clone https://github.com/ggerganov/llama.cpp.git
```

Enter the source directory of the app.

```
cd llama.cpp
```

Build the app.

```
make -j
```

Or,

```
make -j2/j4/j8 (depending on the actual cores of the CPU on the machine)
```

Now, we will download the model files. Here's a simple analogy, model runners are like audio players whereas the model files (LLMs) are like MP3 files. We need files to run the program.

```
cd models
```

```
wget https://huggingface.co/TheBloke/Llama-2-7b-Chat-GGUF/resolve/main/llama-2-7b-chat.Q2_K.gguf
```

```
wget https://huggingface.co/TheBloke/CodeLlama-7B-Instruct-GGUF/resolve/main/codellama-7b-instruct.Q2_K.gguf
```

Switch to the source directory (move back to one directory level). The compiled program binary files are placed in the source directory for this program, "llama.cpp".

```
cd ../
```

Test the setup.

```
./main -m models/llama-2-7b-chat.Q2_K.gguf -p "in pure C programming language, write a function to create a Fibonacci sequence" -n 400 -e
```

```
./main -m models/codellama-7b-instruct.Q2_K.gguf -p "in pure C programming language, write a function to create a Fibonacci sequence" -n 400 -e
```

```
./main -m models/codellama-7b-instruct.Q2_K.gguf -p "in pure C programming language, write a program to create a Fibonacci sequence" -n 400 -e
```

Expect to get satisfactory answers from the AI Chatbot.

Interactive chat:

```
./main  -m models/llama-2-7b-chat.Q2_K.gguf --color \
       --ctx_size 2048 \
       -n -1 \
       -ins -b 256 \
       --top_k 10000 \
       --temp 0.2 \
       --repeat_penalty 1.1
```

```
./main  -m models/codellama-7b-instruct.Q2_K.gguf --color \
       --ctx_size 2048 \
       -n -1 \
       -ins -b 256 \
       --top_k 10000 \
       --temp 0.2 \
       --repeat_penalty 1.1
```

Create bash shell scripts and desktop entry files for easy access.

NOTE: Replace `YOUR_USERNAME` with your actual username, `echo $(whoami)`.

```
echo $(whoami)
```

Find the following files in the folder `Tulu-C-IDE/Artificial-Intelligence-CLI/llama.cpp`. Open the files with a text editor and make the necessary changes.

The launcher scripts (bash):

`interactive-codellama-instruct.sh`

```
#!/bin/bash

cd /home/YOUR_USERNAME/PortablePrograms/llama.cpp \

./main  -m models/codellama-7b-instruct.Q2_K.gguf --color \
       --ctx_size 2048 \
       -n -1 \
       -ins -b 256 \
       --top_k 10000 \
       --temp 0.2 \
       --repeat_penalty 1.1
```

`interactive-llama-chat.sh`

```bash
#!/bin/bash

cd /home/YOUR_USERNAME/PortablePrograms/llama.cpp \

./main  -m models/llama-2-7b-chat.Q2_K.gguf --color \
       --ctx_size 2048 \
       -n -1 \
       -ins -b 256 \
       --top_k 10000 \
       --temp 0.2 \
       --repeat_penalty 1.1
```

Desktop entries for easy access:

`llama-chat.desktop`

```ini
[Desktop Entry]
Type=Application
Name=llama-chat
Comment=AI Chat LLaMA
Icon=fbmessenger
Exec=sakura -e /home/YOUR_USERNAME/PortablePrograms/llama.cpp/interactive-llama-chat.sh
Terminal=false
Categories=;
```

`codellama.desktop`

```ini
[Desktop Entry]
Type=Application
Name=codellama
Comment=AI Chat LLaMA Code Instruct
Icon=fbmessenger
Exec=sakura -e /home/YOUR_USERNAME/PortablePrograms/llama.cpp/interactive-codellama-instruct.sh
Terminal=false
Categories=;
```

Copy the two desktop entry files that you've created to the `~/.local/share/applications` directory. Your launcher should show the entries. Type the following into the address bar of your file manager to enter the `~/.local/share/applications` directory.

```
~/.local/share/applications
```

WizardCoder-Python

If you have a more powerful machine, like a Core i5 4th-gen, give WizardCoder-Python a try.

https://github.com/nlpxucan/WizardLM/blob/main/WizardCoder/README.md

https://www.reddit.com/r/WizardCoder/comments/164xemb/20230826_we_released_wizardcoderpython34bv10/

The 13B-Q2_K GGUF version can be downloaded from here:

https://huggingface.co/TheBloke/WizardCoder-Python-13B-V1.0-GGUF/tree/main

```
wget https://huggingface.co/TheBloke/WizardCoder-Python-13B-V1.0-
GGUF/resolve/main/wizardcoder-python-13b-v1.0.Q2_K.gguf
```

A 34B-Q2_K version of WizardCoder-Python is available. Download that if you have a capable machine to run such a large model file.

https://huggingface.co/TheBloke/WizardCoder-Python-34B-V1.0-GGUF/tree/main

```
wget https://huggingface.co/TheBloke/WizardCoder-Python-34B-V1.0-
GGUF/resolve/main/wizardcoder-python-34b-v1.0.Q2_K.gguf
```

The 13B version of CodeLlama:

```
wget https://huggingface.co/TheBloke/CodeLlama-13B-Instruct-
GGUF/resolve/main/codellama-13b-instruct.Q2_K.gguf
```

Additional AI Models Available for Download

In addition to the conversational AI models described previously, a variety of other AI models are available for download. These models are typically derived from the base models and are therefore not as performant as the conversational AI models. However, they offer a wider range of capabilities, and many of them have been fine-tuned by individuals and small teams, rather than large corporations.

These models are able to generate code, but it is important to be specific in your queries and to break them down into step-by-step instructions, each step of which should be provided in complete detail. For chat-style conversations, it is recommended to choose the chat/instruction versions of AI models.

Marcoroni-7b-GGUF

```
wget https://huggingface.co/TheBloke/Marcoroni-7b-GGUF/resolve/main/marcoroni-7b.Q2_K.gguf
```

tulu-7B-GGUF

```
wget https://huggingface.co/TheBloke/tulu-7B-GGUF/resolve/main/tulu-7b.Q2_K.gguf
```

Mistral-7B-v0.1-GGUF

```
wget https://huggingface.co/TheBloke/Mistral-7B-v0.1-GGUF/resolve/main/mistral-7b-v0.1.Q2_K.gguf
```

gorilla-7B-GGUF

```
wget https://huggingface.co/TheBloke/gorilla-7B-GGUF/resolve/main/Gorilla-7B.Q2_K.gguf
```

openllama-3b-v2-gguf

```
wget https://huggingface.co/klosax/openllama-3b-v2-gguf/resolve/main/openllama-3b-v2-q4_0.gguf
```

Luna-AI-Llama2-Uncensored-GGUF

```
wget https://huggingface.co/TheBloke/Luna-AI-Llama2-Uncensored-GGUF/resolve/main/luna-ai-llama2-uncensored.Q2_K.gguf
```

Guanaco-7B-Uncensored-GGUF

```
wget https://huggingface.co/TheBloke/Guanaco-7B-Uncensored-GGUF/resolve/main/guanaco-7b-uncensored.Q2_K.gguf
```

TheBloke/WizardCoder-Python-7B-V1.0-GGUF · Hugging Face

```
wget https://huggingface.co/TheBloke/WizardCoder-Python-7B-V1.0-GGUF/resolve/main/wizardcoder-python-7b-v1.0.Q2_K.gguf
```

Aryanne/Wizard-Orca-3B-gguf · Hugging Face

```
wget https://huggingface.co/Aryanne/Wizard-Orca-3B-gguf/resolve/main/q4_0-wizard-orca-3b.gguf
```

alexedelsburg/Puma-3b-GGUF · Hugging Face

```
wget https://huggingface.co/alexedelsburg/Puma-3b-GGUF/resolve/main/puma-3b.q4_1.gguf
```

Aryanne/Bean-3B-ggml-and-gguf · Hugging Face

```
wget https://huggingface.co/Aryanne/Bean-3B-ggml-and-gguf/resolve/main/bean-3B-ggufv2-q4_0.gguf
```

Aryanne/Calypso-3B-alpha-v2-gguf · Hugging Face

```
wget
```

Aryanne/TinyLlama-1.1B-Chat-v0.3-gguf · Hugging Face

```
wget https://huggingface.co/Aryanne/TinyLlama-1.1B-Chat-v0.3-gguf/resolve/main/q4_0-tinyllama-1.1b-chat-v0.3.gguf
```

Aryanne/TinyLlama-1.1B-1T-OpenOrca-gguf · Hugging Face

```
wget https://huggingface.co/Aryanne/TinyLlama-1.1B-1T-OpenOrca-gguf/resolve/main/q4_0-tinyllama-1.1b-1t-openorca.gguf
```

TheBloke/dolphin-2.1-mistral-7B-GGUF at main

```
wget https://huggingface.co/TheBloke/dolphin-2.1-mistral-7B-GGUF/resolve/main/dolphin-2.1-mistral-7b.Q2_K.gguf
```

Nomic AI - GPT4All - mistral-7b-instruct-v0.1.Q4_0.gguf

```
wget https://gpt4all.io/models/gguf/mistral-7b-instruct-v0.1.Q4_0.gguf
```

LLaMA.CPP for Microsoft Windows

LLaMA.CPP runs on MS Windows. An appropriate version for Windows 10 can be downloaded from their GitHub release page: https://github.com/ggerganov/llama.cpp/releases

Download non-AVX versions if your CPU does not support AVX/AVX2. Download AVX2 for AVX2-supported CPUs. On my Windows machine, the archive `llama-bxxxx-bin-win-avx2-x64.zip` was the right option.

Make sure you download the `models` directory separately, `llama.cpp/models`. The folder `models` contains some configuration files required by `LLaMA.CPP`.

Multiline text input in LLaMA:

When you run LLaMA.CPP in interactive mode, you will not be able to hit Enter to break your input.

```
== Running in interactive mode. ==
 - Press Ctrl+C to interject at any time.
 - Press Return to return control to LLaMa.
 - To return control without starting a new line, end your input with '/'.
 - If you want to submit another line, end your input with '\'.
```

You'll have to add the string `\` to continue to the subsequent lines. Doing that manually is a daunting task. Initially, I thought I would write two separate C programs using Win32 and GTK+ 3 to do that, with the plan of merging the programs into a single one employing a preprocessor directive. However, a simple web app can serve the purpose. I don't understand HTML/Javascript/CSS, to be honest. So, I asked `wizardcoder-python-13b-v1.0.Q2_K.gguf` to give me a solution. It worked, although the code had minor glitches that were later ironed out by ChatGPT.

If you need to insert multiple lines to LLaMA.CPP, open `formatted-copy.html` in your preferred browser. Type your prompts there. Format your prompts. Then, press a button to add `\` at the end of each line, so you can copy the text in the exact format required by the runner, LLaMA.CPP.

NOTE:

1. Too **high CPU/RAM usage**. Jokes apart, be prepared to heat up your room with your computer.

2. Responses are generated very slowly, nearly one word per second in my setup. These two runner apps do not support GPUs. If they had support for at least the integrated graphics processor of the CPU (e.g., the iGPU in Intel CPUs), generating responses would have been way faster.

3. Requires a reasonably capable machine; at least 16GB DDR3 RAM and a 2GHz quad-core CPU. That requirement is the real minimum.

4. The models cannot be integrated into commercial chatbots, like AI chatbots in customer care services. Most of these models are derivative works of Meta's LLaMA model. Some of them are fine-tuned using OpenAI's responses.

5. For a trouble-free experience, use GPT4ALL in its default settings. It is the best overall model runner around.

Find a detailed review of AI models that you can run locally with GPT4ALL, `ai-model-test.txt`. The review is left as it is with no spelling/grammatical inconsistency correction.