Vicuna-13B: Best Free ChatGPT Alternative According to GPT-4 🤯 | Tutorial (GPU)

Martin Thissen

Wow, in my last article I already showed you how to set up the Vicuna model on your local computer, but the results were not as good as expected.

For this reason, I created a fork and basically merged two repositories to get the Vicuna model up and running, and what can I say, the responses and especially the quality of the responses are insane. Forget alpaca, seriously! You don't believe me? Follow the steps in this article and convince yourself.

If you like videos more, feel free to check out my YouTube video to this article:

You don't have your own GPU? Don't worry, I have also created a Colab notebook that you can run to interact with the model. To understand why the model can run in a Colab notebook, make sure to have a look at the section where I show details about the GPTQ quantization technique.

Foundation: Install Conda

It is always recommended to use a clean and new environment for each project. So if you don't work with Anaconda yet, feel free to install it as your friend for dealing with different Python environments:

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
sha256sum Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
source ~/.bashrc
```

Once Anaconda is installed, we can create a virtual environment for this project:

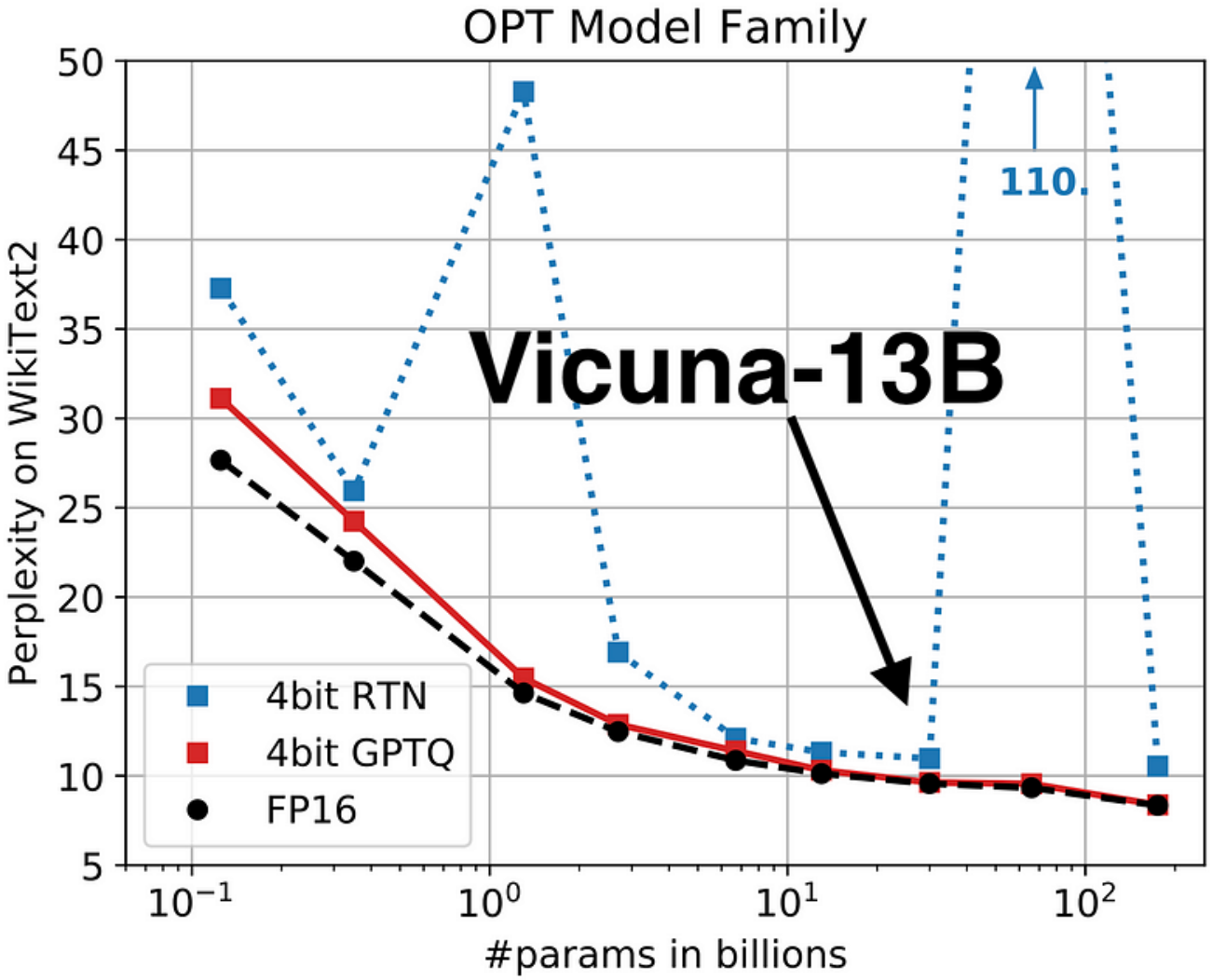
```
conda create -n vicuna python=3.9
conda activate vicuna
```

Installation of the Vicuna model

Okay, now let's move on to the fun part. First, we will clone the forked repository:

```
git clone https://github.com/thisserand/FastChat.git
cd FastChat
pip3 install -e .
```

Since the FastChat (Vicuna) repository doesn't yet support GPTQ-quantized models, I have integrated the GPTQ-for-LLaMa repository into this fork to run a GPTQ-quantized model. Why should we use the GPTQ-quantized version of the Vicuna model? By using the GPTQ-quantized version, we can reduce the VRAM requirement from 28 GB to about 10 GB, which allows us to run the Vicuna-13B model on a single consumer GPU. Another advantage is the ~3x speedup during inference when using the quantized model version. Any disadvantages? Yes, the quantized model is slightly less precise and therefore performs slightly worse than the full precision model. However, it is the quantization of the model that allows many of us to use the model at all. The following figure from the GPTQ paper shows that, especially for larger language models, the qualitative degradation due to the application of quantization is small:



Original figure is from the GPTQ paper.

Okay, back to the installation. Make sure you clone the GPTQ-for-LLaMa repository to the repositories folder (and not somewhere else). I ran this code in a cloud GPU instance and initially ran into a problem because the CUDA devtools (nvcc) were not installed. So I changed the Docker image I was using to *nvidia/cuda:11.7.0-devel-ubuntu18.04* to resolve this issue. If you have similar problems, either install the

cuda-devtools or change the image as well.

```
mkdir repositories
cd repositories
git clone https://github.com/oobabooga/GPTQ-for-LLaMa.git -b cuda
cd GPTQ-for-LLaMa
python setup_cuda.py install
```

Once this is done, we navigate back to our main folder (FastChat) and download the GPTQ-quantized Vicuna model:

```
cd ../..
python download-model.py anon8231489123/vicuna-13b-GPTQ-4bit-128g
```

And that’s it, now we can use the Vicuna model. There are two ways for this: either interacting with the model via the CLI (e.g. terminal or cmd) or via the web UI (which I prefer).

CLI Usage

Using the command line interface (CLI) is pretty straightforward, just run the following command. Have fun exploring the possibilities of the Vicuna model :-)

```
python -m fastchat.serve.cli --model-name anon8231489123/vicuna-13b-GPTQ-4bit-128g --wbits 4 --groupsize 128
```

Web UI Usage

Using the Web UI is a little more complicated, but don’t worry. Overall, we need to run three different things: the controller, the model_worker, and the web UI. As you can imagine, we could theoretically run multiple model_workers in this setup, but let’s just start with a single model_worker. One other thing I’d like to mention is that my Docker image didn’t have IPv6 set up, forcing me to pass the location of the various servers as an argument to each command. Feel free to leave them out if IPv6 is working in your environment.

Since I was using the model in a GPU cloud instance, I could not open multiple terminals and had to start all three different servers in one terminal session. For this reason, I first had to install *screen*, which can be used to open any number of virtual terminals. Therefore, I have divided the next step into two sections. The first is for everyone who also needs to use *screen* and can’t open multiple terminals. The second is for all who work on their local machine and can open multiple terminals.

Option 1: Using Screen (to Use Virtual Terminals)