exl2 quantization for dummies

Tutorial | Guide

Given the recent disappearance of a formerly very prolific releaser of quantized models, I thought I would try to come up with a workflow for users to quantize their own models with the absolute minimum of setup. Special thanks to u/Knopty for help in debugging this workflow.

For this tutorial I have chosen exllamav2's exl2 format as it is both performant and allows users to pick their own bits per weight (including fractional values), to optimize a model for their VRAM budget.

To manage the majority of the requirements I am going to use oobabooga's text generation UI one-click installation and assume some familiarity with its UI (loading a model and running inference in the chat is sufficient).

1. Download and install the latest textgen ui release from the repository and run the appropriate script for your OS. I shall assume a native Windows installation for this tutorial. Follow the prompts given by oobabooga and launch it from a browser as described in the textgen UI readme.md. If it is already installed, run the update script for your OS (e.g. native Windows users would run update_windows.bat).
2. I recommend you start textgen UI from the appropriate start script and test that you can load exl2 models before continuing further by downloading a pre-quantized model from HuggingFace and loading it with the exllamav2_HF loader. For Windows this would be start_windows.bat. exllamav2's author has some examples on his HF page, each is available in a variety of bpws selected from the branch dropdown (that's the *main* button on the *files and versions* tab of the model page for those unfamiliar with Git).
3. Locate the unquantized (FP16/BF16) model you wish to quantize on Hugging Face. These can usually be identified by the lack of any quantization format in the title. For this example I shall use open_llama_3b, I suggest for your first attempt you also choose a small model whose unquantized version is small enough to fit in your VRAM, in this case the unquantized model is 6.5GB. Download all the files from the hugging face repository to your oobabooga models folder (text-generation-webui\models), if you are feeling masochistic you can have the webui do this for you.
4. Locate the cmd_ script for your operating system in the text-generation-webui folder. E.g. I shall run cmd_windows.bat. This will activate ooba's a conda environment giving you access to all the dependencies that exllamv2 will need for quantization.
5. This particular model is in pickle format (.bin) and for quantization we need this in .safetensors format, so we shall first need to convert it. If your selected model is already in .safetensors format then skip to step 7. Otherwise in the conda terminal, enter `python convert-to-safetensors.py input_path -o output_path` where input_path is the folder containing your .bin pickle and the output_path is a folder to store the safetensors version of the model. E.g. `python convert-to-safetensors.py models/openlm-research_open_llama_3b -o models/open_llama_3b_fp16`
6. Once the safetensors model is finished you may wish to load it in the textgen UI's *Transfomers* loader to confirm you have succeded in converting the model. Just make sure you unload the model before continuing further.
7. Now we need a release of exllamav2 containing the convert.py quantization script. This is currently not included in the textgen UI pip package, so you will need a separate copy of exllamav2. I recommend you download the latest version from the

repository's releases page as this needs to match with the dependencies that textgen UI has installed. For this tutorial I shall download the Source Code.zip for 0.0.13post2 and unzip it into the textgeneration-webui folder (it doesn't need to be in here, but the path should not contain spaces). So in my case this is text-generation-webui\exllamav2-0.0.13.post2 I'm also going to create a folder called *working* inside this folder to hold temporary files during the quantization which I can discard when it's finished.

8. In the conda terminal, change directory to the exllamav2 folder, e.g. `cd exllamav2-0.0.13.post2`
9. Exllamav2 uses a measurement based quantization method, whereby it measures the errors introduced by quantization and attempts to allocate the available bpw budget intelligently to those weights that have the most impact on the performance of the model. To do these measurements the quantizer will run inference of some calibration data and evaluate the losses for different bits per weight. In this example we are going to use exllamav2's internal calibration dataset which should be sufficient for less agressive quantizations and a more general use case. For aggressive quants (<4 bpw) and niche use cases, it is recommended you use a custom data set suited to your end use. Many of these can be found as datasets on HuggingFace. The dataset needs to be in .parquet format. If you do use a custom calibration file, you will need to specify its path using the -c argument in the next step.
10. Now we are ready to quantize! I suggest you monitor your RAM and VRAM usage during this step to see if you are running out of memory (which will cause quantization speed to drop dramatically), Windows users can do this from the performance tab of task manager. In the conda terminal enter `python convert.py -i input_path -o working_path -cf output_path -hb head_size -b bpw`. -b is the BPW for the majority of the layers, head_size is the bpw of the output layer which should be either 6 or 8 (for b>=6 I recommend hb=8 else hb=6). So in this example my models are in the text-generation-webui\models folder so I shall use: `python convert.py -i ../models/open_llama_3b_fp16 -o working -cf ../models/open_llama_3b_exl2 -b 6 -hb 8 -nr` The -nr flag here is just flushing the working folder of files before starting a new job.
11. The quantization should now start with the measurement pass then run the quantization itself. For me quantizing this 3B model on an RTX 4060 Ti 16GB, the measurement pass used 3.6GB of RAM, 2.8GB of VRAM and took about eight minutes, the quantization itself used 6GB of RAM and 3.2GB of VRAM and took seven minutes. Obviously larger models will require more resources to quantize.
12. Load your newly quantized exl2 in the textgen UI and enjoy.

Give a man a pre-quantized model and you feed him for a day before he asks you for another quant for a slightly different but supposedly superior merge. Teach a man to quant and he feeds himself with his own compute.