bartowski / granite-3.0-2b-instruct-GGUF

like 2

Text Generation | GGUF | language | granite-3.0 | Eval Results | imatrix | conversational | License: apache-2.0

Use this model

Model card | Files | Community

Downloads last month
0

GGUF

Model size 2.63B params | Architecture granite

2-bit	Q2_K, Q2_K_L
3-bit	IQ3_XXS, Q3_K_S, Q3_K_L, Q3_K_XL
4-bit	IQ4_XS, Q4_K_S, Q4_0, Q4_0, Q4_0, Q4_0, Q4_K_M, Q4_K_L
5-bit	Q5_K_S, Q5_K_M, Q5_K_L
6-bit	Q6_K, Q6_K_L
8-bit	Q8_0
16-bit	F16

View +2 files

Inference Examples

Text Generation

Inference API (serverless) has been turned off for this model.

Model tree for bartowski/granite-3.0-2b-instruct-GGUF

Base model ibm-granite/granite-3.0-2b-instruct

Quantized (14) ..... this model

Evaluation results

pass@1 on IFEval	self-reported	46.070
pass@1 on IFEval	self-reported	7.660
pass@1 on AGI-Eval	self-reported	29.750
pass@1 on AGI-Eval	self-reported	56.030
pass@1 on AGI-Eval	self-reported	27.920
pass@1 on OBQA	self-reported	43.200
pass@1 on OBQA	self-reported	66.360
pass@1 on OBQA	self-reported	76.790
pass@1 on OBQA	self-reported	71.900
pass@1 on OBQA	self-reported	53.370

Expand 23 evaluations

View on Papers With Code

Edit model card

Llamacpp imatrix Quantizations of granite-3.0-2b-instruct

Using llama.cpp release b3930 for quantization.

Original model: https://huggingface.co/ibm-granite/granite-3.0-2b-instruct

All quants made using imatrix option with dataset from here

Run them in LM Studio

Prompt format

```
<|start_of_role|>system<|end_of_role|>{system_prompt}<|end_of_text|>
<|start_of_role|>user<|end_of_role|>{prompt}<|end_of_text|>
<|start_of_role|>assistant<|end_of_role|>
```

Download a file (not the whole branch) from below:

Filename	Quant type	File Size	Split	Description
granite-3.0-2b-instruct-f16.gguf	f16	5.27GB	false	Full F16 weights.
granite-3.0-2b-instruct-Q8_0.gguf	Q8_0	2.80GB	false	Extremely high quality, generally unneeded but max available quant.
granite-3.0-2b-instruct-Q6_K_L.gguf	Q6_K_L	2.21GB	false	Uses Q8_0 for embed and output weights. Very high quality, near perfect, *recommended*.
granite-3.0-2b-instruct-Q6_K.gguf	Q6_K	2.16GB	false	Very high quality, near perfect, *recommended*.
granite-3.0-2b-instruct-Q5_K_L.gguf	Q5_K_L	1.94GB	false	Uses Q8_0 for embed and output weights. High quality, *recommended*.
granite-3.0-2b-instruct-Q5_K_M.gguf	Q5_K_M	1.87GB	false	High quality, *recommended*.
granite-3.0-2b-instruct-Q5_K_S.gguf	Q5_K_S	1.83GB	false	High quality, *recommended*.
granite-3.0-2b-instruct-Q4_K_L.gguf	Q4_K_L	1.68GB	false	Uses Q8_0 for embed and output weights. Good quality, *recommended*.
granite-3.0-2b-instruct-Q4_K_M.gguf	Q4_K_M	1.60GB	false	Good quality, default size for must use cases, *recommended*.
granite-3.0-2b-instruct-Q4_K_S.gguf	Q4_K_S	1.52GB	false	Slightly lower quality with more space savings, *recommended*.
granite-3.0-2b-instruct-Q4_0.gguf	Q4_0	1.52GB	false	Legacy format, generally not worth using over similarly sized formats
granite-3.0-2b-instruct-Q4_0_8_8.gguf	Q4_0_8_8	1.51GB	false	Optimized for ARM inference. Requires 'sve' support (see link below). *Don't use on Mac or Windows*.
granite-3.0-2b-instruct-Q4_0_4_8.gguf	Q4_0_4_8	1.51GB	false	Optimized for ARM inference. Requires 'i8mm' support (see link below). *Don't use on Mac or Windows*.
granite-3.0-2b-instruct-Q4_0_4_4.gguf	Q4_0_4_4	1.51GB	false	Optimized for ARM inference. Should work well on all ARM chips, pick this if you're unsure. *Don't use on Mac or Windows*.
granite-3.0-2b-instruct-Q3_K_XL.gguf	Q3_K_XL	1.49GB	false	Uses Q8_0 for embed and output weights. Lower quality but usable, good for low RAM availability.
granite-3.0-2b-instruct-IQ4_XS.gguf	IQ4_XS	1.44GB	false	Decent quality, smaller than Q4_K_S with similar performance, *recommended*.
granite-3.0-2b-instruct-Q3_K_L.gguf	Q3_K_L	1.40GB	false	Lower quality but usable, good for low RAM availability.
granite-3.0-2b-instruct-IQ3_M.gguf	IQ3_M	1.21GB	false	Medium-low quality, new method with decent performance comparable to Q3_K_M.
granite-3.0-2b-instruct-Q3_K_S.gguf	Q3_K_S	1.17GB	false	Low quality, not recommended.
granite-3.0-2b-instruct-IQ3_XS.gguf	IQ3_XS	1.12GB	false	Lower quality, new method with decent performance, slightly better than Q3_K_S.
granite-3.0-2b-instruct-Q2_K_L.gguf	Q2_K_L	1.11GB	false	Uses Q8_0 for embed and output weights. Very low quality but surprisingly usable.
granite-3.0-2b-instruct-IQ3_XXS.gguf	IQ3_XXS	1.05GB	false	Lower quality, new method with decent performance, comparable to Q3 quants.

Filename	Quant type	File Size	Split	Description
granite-3.0-2b-instruct-Q2_K.gguf	Q2_K	1.01GB	false	Very low quality but surprisingly usable.

Embed/output weights

Some of these quants (Q3_K_XL, Q4_K_L etc) are the standard quantization method with the embeddings and output weights quantized to Q8_0 instead of what they would normally default to.

Some say that this improves the quality, others don't notice any difference. If you use these models PLEASE COMMENT with your findings. I would like feedback that these are actually used and useful so I don't keep uploading quants no one is using.

Thanks!

Downloading using huggingface-cli

First, make sure you have hugginface-cli installed:

```
pip install -U "huggingface_hub[cli]"
```

Then, you can target the specific file you want:

```
huggingface-cli download bartowski/granite-3.0-2b-instruct-GGUF --inclu
```

If the model is bigger than 50GB, it will have been split into multiple files. In order to download them all to a local folder, run:

```
huggingface-cli download bartowski/granite-3.0-2b-instruct-GGUF --inclu
```

You can either specify a new local-dir (granite-3.0-2b-instruct-Q8_0) or download them all in place (./)

Q4_0_X_X

These are *NOT* for Metal (Apple) offloading, only ARM chips.

If you're using an ARM chip, the Q4_0_X_X quants will have a substantial speedup. Check out Q4_0_4_4 speed comparisons on the original pull request

To check which one would work best for your ARM chip, you can check AArch64 SoC features (thanks EloyOn!).

Which file should I choose?

A great write up with charts showing various performances is provided by Artefact2 here

The first thing to figure out is how big a model you can run. To do this, you'll need to figure out how much RAM and/or VRAM you have.

If you want your model running as FAST as possible, you'll want to fit the whole thing on your GPU's VRAM. Aim for a quant with a file size 1-2GB smaller than your GPU's total VRAM.

If you want the absolute maximum quality, add both your system RAM and your GPU's VRAM together, then similarly grab a quant with a file size 1-2GB Smaller than that total.

Next, you'll need to decide if you want to use an 'I-quant' or a 'K-quant'.

If you don't want to think too much, grab one of the K-quants. These are in format 'QX_K_X', like Q5_K_M.

If you want to get more into the weeds, you can check out this extremely useful feature chart:

llama.cpp feature matrix

But basically, if you're aiming for below Q4, and you're running cuBLAS (Nvidia) or rocBLAS (AMD), you should look towards the I-quants. These are in format IQX_X, like IQ3_M. These are newer and offer better performance for their size.

These I-quants can also be used on CPU and Apple Metal, but will be slower than their K-quant equivalent, so speed vs performance is a tradeoff you'll have to decide.

The I-quants are *not* compatible with Vulcan, which is also AMD, so if you have an AMD card double check if you're using the rocBLAS build or the Vulcan build. At the time of writing this, LM Studio has a preview with ROCm support, and other inference engines have specific builds for ROCm.

Credits

Thank you kalomaze and Dampf for assistance in creating the imatrix calibration dataset

Thank you ZeroWw for the inspiration to experiment with embed/output

Want to support my work? Visit my ko-fi page here: https://ko-fi.com/bartowski

---