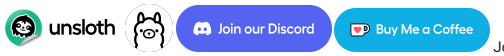
To run this, press "*Runtime*" and press "*Run all*" on a **free** Tesla T4 Google Colab instance!



Join Discord if you need help + ⭐ *Star us on Github* ⭐

To install Unsloth on your own computer, follow the installation instructions on our Github page here.

You will learn how to do data prep and import a CSV, how to train, how to run the model, & how to export to Ollama!

Unsloth now allows you to automatically finetune and create a Modelfile, and export to Ollama! This makes finetuning much easier and provides a seamless workflow from `Unsloth` to `Ollama`! We now allow uploading CSVs, Excel files!

We'll be using the Titanic dataset in a CSV to try predict with a finetuned model who survived or perished.

```
%%capture
# Installs Unsloth, Xformers (Flash Attention) and all other packages!
!pip install unsloth
# Get latest Unsloth
!pip install --upgrade --no-deps "unsloth[colab-new] @ git+https://github.com/unslothai/unsloth.git"
```

- We support Llama, Mistral, Phi-3, Gemma, Yi, DeepSeek, Qwen, TinyLlama, Vicuna, Open Hermes etc
- We support 16bit LoRA or 4bit QLoRA. Both 2x faster.
- `max_seq_length` can be set to anything, since we do automatic RoPE Scaling via kaiokendev's method.
- With PR 26037, we support downloading 4bit models **4x faster**! Our repo has Llama, Mistral 4bit models.
- [**NEW**] We make Phi-3 Medium / Mini **2x faster**! See our Phi-3 Medium notebook

```
from unsloth import FastLanguageModel
import torch
max_seq_length = 2048 # Choose any! We auto support RoPE Scaling internally!
dtype = None # None for auto detection. Float16 for Tesla T4, V100, Bfloat16 for Ampere+
load_in_4bit = True # Use 4bit quantization to reduce memory usage. Can be False.

# 4bit pre quantized models we support for 4x faster downloading + no OOMs.
fourbit_models = [
    "unsloth/mistral-7b-v0.3-bnb-4bit",      # New Mistral v3 2x faster!
    "unsloth/mistral-7b-instruct-v0.3-bnb-4bit",
    "unsloth/llama-3-8b-bnb-4bit",           # Llama-3 15 trillion tokens model 2x faster!
    "unsloth/llama-3-8b-Instruct-bnb-4bit",
    "unsloth/llama-3-70b-bnb-4bit",
    "unsloth/Phi-3-mini-4k-instruct",        # Phi-3 2x faster!
    "unsloth/Phi-3-medium-4k-instruct",
    "unsloth/mistral-7b-bnb-4bit",
    "unsloth/gemma-7b-bnb-4bit",             # Gemma 2.2x faster!
] # More models at https://huggingface.co/unsloth

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/llama-3-8b-bnb-4bit",
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
    # token = "hf_...", # use one if using gated models like meta-llama/Llama-2-7b-hf
)
```

```
🦥 Unsloth: Will patch your computer to enable 2x faster free finetuning.
config.json: 100%                                                 1.20k/1.20k [00:00<00:00, 30.7kB/s]
==((====))==  Unsloth: Fast Llama patching release 2024.6
   \\   /|    GPU: Tesla T4. Max memory: 14.748 GB. Platform = Linux.
O^O/ \_/ \    Pytorch: 2.3.0+cu121. CUDA = 7.5. CUDA Toolkit = 12.1.
\        /    Bfloat16 = FALSE. Xformers = 0.0.26.post1. FA = False.
 "-____-"     Free Apache license: http://github.com/unslothai/unsloth
model.safetensors: 100%                                           5.70G/5.70G [00:30<00:00, 192MB/s]
generation_config.json: 100%                                      172/172 [00:00<00:00, 12.8kB/s]
tokenizer_config.json: 100%                                       50.6k/50.6k [00:00<00:00, 2.25MB/s]
tokenizer.json: 100%                                              9.09M/9.09M [00:00<00:00, 14.2MB/s]
special_tokens_map.json: 100%                                     464/464 [00:00<00:00, 35.4kB/s]
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
```

We now add LoRA adapters so we only need to update 1 to 10% of all parameters!

```
model = FastLanguageModel.get_peft_model(
    model,
    r = 16, # Choose any number > 0 ! Suggested 8, 16, 32, 64, 128
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                      "gate_proj", "up_proj", "down_proj",],
    lora_alpha = 16,
    lora_dropout = 0, # Supports any, but = 0 is optimized
    bias = "none",    # Supports any, but = "none" is optimized
    # [NEW] "unsloth" uses 30% less VRAM, fits 2x larger batch sizes!
    use_gradient_checkpointing = "unsloth", # True or "unsloth" for very long context
    random_state = 3407,
    use_rslora = False,  # We support rank stabilized LoRA
    loftq_config = None, # And LoftQ
)
```

```
Unsloth 2024.6 patched 32 layers with 32 QKV layers, 32 O layers and 32 MLP layers.
```

Data Prep

We'll now use the Titanic dataset, which is a CSV / Excel file with many columns. The goal is to predict whether some passengers managed to survive or perish based on their characteristics like their age, how much was their fare etc.

We uploaded it to our HF repo, but you can upload a CSV by pressing the icon to the left and press the upload button.

```
from datasets import load_dataset
dataset = load_dataset(
    "csv",
    data_files = "https://huggingface.co/datasets/unsloth/datasets/raw/main/titanic.csv",
    split = "train",
)
print(dataset.column_names)
print(dataset[0])
```

```
Downloading data: 100%  61.2k/61.2k [00:00<00:00, 652kB/s]
Generating train split:  891/0 [00:00<00:00, 5529.42 examples/s]
['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked']
{'PassengerId': 1, 'Survived': 0, 'Pclass': 3, 'Name': 'Braund, Mr. Owen Harris', 'Sex': 'male', 'Age': 22.0, 'SibSp': 1, 'Parch': 0, 'Ticket': 'A/5 21171',
```

One issue is this dataset has multiple columns. For `Ollama` and `llama.cpp` to function like a custom `ChatGPT` Chatbot, we must only have 2 columns - an `instruction` and an `output` column.

```
print(dataset.column_names)
```

```
['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked']
```

To solve this, we shall do the following:

- Merge all columns into 1 instruction prompt.
- Remember LLMs are text predictors, so we can customize the instruction to anything we like!
- Use the `to_sharegpt` function to do this column merging process!

PassengerId	Survived	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Sex
1	0	3	Braund, M	22	1	0	A/5 211	7.25		S	male
2	1	1	Cumings,	38	1	0	PC 175	71.28	C85	C	female
3	1	3	Heikkinen	26	0	0	STON/C	7.925		S	female
4	1	1	Futrelle, M	35	1	0	113803	53.1	C123	S	female
5	0	3	Allen, Mr.	35	0	0	373450	8.05		S	male

Merge

They have 1 siblings and spouses.
Their passenger class is 3.
Their age is 22.0.
They paid $7.25 for the trip.

To merge multiple columns into 1, use `merged_prompt`.

- Enclose all columns in curly braces `{}`.
- Optional text must be enclosed in `[[]]`. For example if the column "Pclass" is empty, the merging function will not show the text and skp this. This is useful for datasets with missing values.
- You can select every column, or a few!
- Select the output or target / prediction column in `output_column_name`. For the Titanic dataset, this will be `Survived`.

For example, if we want to use the columns `Age` and `Fare`, we can do the following:

```
from unsloth import to_sharegpt
dataset_simple = to_sharegpt(
    dataset,
    merged_prompt = "[[Their age is {Age}.\n]][[They paid ${Fare} for the trip.\n]]",
    output_column_name = "Survived",
)
```

```
Merging columns: 100%  891/891 [00:00<00:00, 12446.17 examples/s]
Converting to ShareGPT: 100%  891/891 [00:00<00:00, 19530.41 examples/s]
```

We shall now provide a complex example using nearly all the columns in the dataset as shown below!

We also provide a setting called `conversation_extension`. This selects a few random rows in the dataset and combines them into 1 conversation. This allows the custom finetune to now not only work on only 1 user input, but many, allowing it be to a true chatbot like ChatGPT!

```
from unsloth import to_sharegpt
dataset = to_sharegpt(
    dataset,
    merged_prompt = \
        "[[The passenger embarked from {Embarked}.]]"\
        "[[\nThey are {Sex}.]]"\
        "[[\nThey have {Parch} parents and childen.]]"\
        "[[\nThey have {SibSp} siblings and spouses.]]"\
        "[[\nTheir passenger class is {Pclass}.]]"\
        "[[\nTheir age is {Age}.]]"\
        "[[\nThey paid ${Fare} for the trip.]]",
    conversation_extension = 5, # Randomnly combines conversations into 1! Good for long convos
    output_column_name = "Survived",
)
```

```
Flattening the indices: 100%  891/891 [00:00<00:00, 14842.58 examples/s]
Flattening the indices: 100%  891/891 [00:00<00:00, 12084.20 examples/s]
Flattening the indices: 100%  891/891 [00:00<00:00, 12392.52 examples/s]
Flattening the indices: 100%  891/891 [00:00<00:00, 16717.33 examples/s]
Flattening the indices: 100%  891/891 [00:00<00:00, 12138.76 examples/s]
Extending conversations: 100%  891/891 [00:00<00:00, 2305.87 examples/s]
```

Let's print out how the dataset looks like now:

```
from pprint import pprint
pprint(dataset[0])
```

```
{'conversations': [{'content': 'Their age is 22.0.\n'
                               'They paid $7.25 for the trip.\n',
                    'from': 'user'},
                   {'content': '0', 'from': 'assistant'},
                   {'content': 'Their age is 52.0.\n'
                               'They paid $79.65 for the trip.\n',
                    'from': 'user'},
                   {'content': '0', 'from': 'assistant'},
                   {'content': 'Their age is 9.0.\n'
                               'They paid $31.275 for the trip.\n',
                    'from': 'user'},
                   {'content': '0', 'from': 'assistant'},
                   {'content': 'They paid $7.8958 for the trip.\n',
                    'from': 'user'},
                   {'content': '0', 'from': 'assistant'},
                   {'content': 'Their age is 24.0.\n'
                               'They paid $13.0 for the trip.\n',
                    'from': 'user'},
                   {'content': '0', 'from': 'assistant'}]}
```

Finally use `standardize_sharegpt`! It converts all `user`, `assistant` and `system` tags to OpenAI Hugging Face style, since sometimes people use different tags like `human` for the `user` and `gpt` for the `assistant`. We require `user` and `assistant`.

```
from unsloth import standardize_sharegpt
dataset = standardize_sharegpt(dataset)
```

```
Standardizing format: 100%  891/891 [00:00<00:00, 2354.41 examples/s]
```

Customizable Chat Templates

You also need to specify a chat template. Previously, you could use the Alpaca format as shown below.

```
alpaca_prompt = """Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately complete

### Instruction:
{}

### Input:
{}

### Response:
{}"""
```

The issue is the Alpaca format has 3 fields, whilst OpenAI style chatbots must only use 2 fields (instruction and response). That's why we used the `to_sharegpt` function to merge these columns into 1.

- Now, you have to use `{INPUT}` for the instruction and `{OUTPUT}` for the response.

```
chat_template = """Below describes some details about some passengers who went on the Titanic.
Predict whether they survived or perished based on their characteristics.
Output 1 if they survived, and 0 if they died.
>>> Passenger Details:
{INPUT}
>>> Did they survive?
{OUTPUT}"""

from unsloth import apply_chat_template
dataset = apply_chat_template(
    dataset,
    tokenizer = tokenizer,
    chat_template = chat_template,
    # default_system_message = "You are a helpful assistant", << [OPTIONAL]
)
```

```
Unsloth: We automatically added an EOS token to stop endless generations.
Map: 100%  891/891 [00:00<00:00, 1242.55 examples/s]
```

We also allow you to use an optional `{SYSTEM}` field. This is useful for Ollama when you want to use a custom system prompt (also like in ChatGPT).

You can also not put a `{SYSTEM}` field, and just put plain text.

```
chat_template = """{SYSTEM}
USER: {INPUT}
ASSISTANT: {OUTPUT}"""
```

Use below if you want to use the Llama-3 prompt format. You must use the `instruct` and not the `base` model if you use this!

```
chat_template = """<|begin_of_text|><|start_header_id|>system<|end_header_id|>

{SYSTEM}<|eot_id|><|start_header_id|>user<|end_header_id|>

{INPUT}<|eot_id|><|start_header_id|>assistant<|end_header_id|>

{OUTPUT}<|eot_id|>"""
```

For the ChatML format:

```
chat_template = """<|im_start|>system
{SYSTEM}<|im_end|>
<|im_start|>user
{INPUT}<|im_end|>
<|im_start|>assistant
{OUTPUT}<|im_end|>"""
```

Train the model

Now let's use Huggingface TRL's `SFTTrainer`! More docs here: TRL SFT docs. We do 60 steps to speed things up, but you can set `num_train_epochs=1` for a full run, and turn off `max_steps=None`. We also support TRL's `DPOTrainer`!

```
from trl import SFTTrainer
from transformers import TrainingArguments
from unsloth import is_bfloat16_supported

trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = dataset,
    dataset_text_field = "text",
    max_seq_length = max_seq_length,
    dataset_num_proc = 2,
    packing = False, # Can make training 5x faster for short sequences.
    args = TrainingArguments(
        per_device_train_batch_size = 2,
        gradient_accumulation_steps = 4,
        warmup_steps = 5,
        max_steps = 60,
        learning_rate = 2e-4,
        fp16 = not is_bfloat16_supported(),
        bf16 = is_bfloat16_supported(),
        logging_steps = 1,
        optim = "adamw_8bit",
        weight_decay = 0.01,
        lr_scheduler_type = "linear",
        seed = 3407,
        output_dir = "outputs",
    ),
)
```

```
/usr/local/lib/python3.10/dist-packages/multiprocess/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork() is incompatible with multithreaded cod
  self.pid = os.fork()
Map (num_proc=2): 100%                                                  891/891 [00:04<00:00, 234.47 examples/s]
max_steps is given, it will override any value given in num_train_epochs
```

Show current memory stats

Show code

```
GPU = Tesla T4. Max memory = 14.748 GB.
5.594 GB of memory reserved.
```

```
trainer_stats = trainer.train()
```

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs = 1
   \\   /|    Num examples = 891 | Num Epochs = 1
O^O/ \_/ \    Batch size per device = 2 | Gradient Accumulation steps = 4
\        /    Total batch size = 8 | Total steps = 60
 "-____-"     Number of trainable parameters = 41,943,040
```

[60/60 05:59, Epoch 0/1]

Step	Training Loss
1	1.782300
2	1.738900
3	1.728600
4	1.508200
5	1.306400
6	1.119500
7	0.946400
8	0.780200
9	0.662400
10	0.571200
11	0.506700
12	0.429000
13	0.453700
14	0.415700
15	0.356500
16	0.372300
17	0.358700
18	0.366100
19	0.354800
20	0.363200
21	0.332300
22	0.350700
23	0.329000
24	0.337300
25	0.326000
26	0.326700
27	0.322600
28	0.301100
29	0.308500
30	0.336400
31	0.285600
32	0.292300
33	0.307100
34	0.292300
35	0.298500
36	0.293600
37	0.277600
38	0.288500
39	0.284800
40	0.313400
41	0.322400
42	0.310800
43	0.281500
44	0.295900
45	0.265400
46	0.286100
47	0.272000
48	0.284500
49	0.270000
50	0.292600
51	0.289400
52	0.271600
53	0.303800
54	0.264100

55	0.297800
56	0.265500
57	0.273500
58	0.280500
59	0.278700
60	0.267000

Show final memory and time stats

Show code

```
378.001 seconds used for training.
6.3 minutes used for training.
Peak reserved memory = 6.291 GB.
Peak reserved memory for training = 0.697 GB.
Peak reserved memory % of max memory = 42.657 %.
Peak reserved memory for training % of max memory = 4.726 %.
```

Inference

Let's run the model! Unsloth makes inference natively 2x faster as well! You should use prompts which are similar to the ones you had finetuned on, otherwise you might get bad results!

```
FastLanguageModel.for_inference(model) # Enable native 2x faster inference
messages = [                    # Change below!
    {"role": "user", "content": 'The passenger embarked from S.\n'\
                                'They are male.\n'\
                                'They have 1 siblings and spouses.\n'\
                                'Their passenger class is 3.\n'\
                                'Their age is 22.0.\n'\
                                'They paid $7.25 for the trip.'},
]
input_ids = tokenizer.apply_chat_template(
    messages,
    add_generation_prompt = True,
    return_tensors = "pt",
).to("cuda")

from transformers import TextStreamer
text_streamer = TextStreamer(tokenizer, skip_prompt = True)
_ = model.generate(input_ids, streamer = text_streamer, max_new_tokens = 128, pad_token_id = tokenizer.eos_token_id)
```

```
0<|end_of_text|>
```

Let's try another example:

```
FastLanguageModel.for_inference(model) # Enable native 2x faster inference
messages = [                    # Change below!
    {"role": "user", "content": 'Their passenger class is 1.\n'\
                                'Their age is 22.0.\n'\
                                'They paid $107.25 for the trip.'},
]
input_ids = tokenizer.apply_chat_template(
    messages,
    add_generation_prompt = True,
    return_tensors = "pt",
).to("cuda")

from transformers import TextStreamer
text_streamer = TextStreamer(tokenizer, skip_prompt = True)
_ = model.generate(input_ids, streamer = text_streamer, max_new_tokens = 128, pad_token_id = tokenizer.eos_token_id)
```

```
0<|end_of_text|>
```

Saving, loading finetuned models

To save the final model as LoRA adapters, either use Huggingface's `push_to_hub` for an online save or `save_pretrained` for a local save.

[NOTE] This ONLY saves the LoRA adapters, and not the full model. To save to 16bit or GGUF, scroll down!

```
model.save_pretrained("lora_model") # Local saving
tokenizer.save_pretrained("lora_model")
# model.push_to_hub("your_name/lora_model", token = "...") # Online saving
# tokenizer.push_to_hub("your_name/lora_model", token = "...") # Online saving
```

```
('lora_model/tokenizer_config.json',
 'lora_model/special_tokens_map.json',
 'lora_model/tokenizer.json')
```

Now if you want to load the LoRA adapters we just saved for inference, set `False` to `True`:

```
if False:
    from unsloth import FastLanguageModel
```

```
    model, tokenizer = FastLanguageModel.from_pretrained(
        model_name = "lora_model", # YOUR MODEL YOU USED FOR TRAINING
        max_seq_length = max_seq_length,
        dtype = dtype,
        load_in_4bit = load_in_4bit,
    )
    FastLanguageModel.for_inference(model) # Enable native 2x faster inference
pass

messages = [                    # Change below!
    {"role": "user", "content": 'Their passenger class is 3.\n'\
                                'Their age is 22.0.\n'\
                                'They paid $107.25 for the trip.'},
]
input_ids = tokenizer.apply_chat_template(
    messages,
    add_generation_prompt = True,
    return_tensors = "pt",
).to("cuda")

from transformers import TextStreamer
text_streamer = TextStreamer(tokenizer, skip_prompt = True)
_ = model.generate(input_ids, streamer = text_streamer, max_new_tokens = 128, pad_token_id = tokenizer.eos_token_id)
```

```
0<|end_of_text|>
```

You can also use Hugging Face's `AutoModelForPeftCausalLM`. Only use this if you do not have `unsloth` installed. It can be hopelessly slow, since `4bit` model downloading is not supported, and Unsloth's **inference is 2x faster**.

```
if False:
    # I highly do NOT suggest - use Unsloth if possible
    from peft import AutoPeftModelForCausalLM
    from transformers import AutoTokenizer
    model = AutoPeftModelForCausalLM.from_pretrained(
        "lora_model", # YOUR MODEL YOU USED FOR TRAINING
        load_in_4bit = load_in_4bit,
    )
    tokenizer = AutoTokenizer.from_pretrained("lora_model")
```

Ollama Support

Unsloth now allows you to automatically finetune and create a Modelfile, and export to Ollama! This makes finetuning much easier and provides a seamless workflow from `Unsloth` to `Ollama`!

Let's first install `Ollama`!

```
!curl -fsSL https://ollama.com/install.sh | sh
```

```
>>> Downloading ollama...
######################################################################################################## 100.0%
>>> Installing ollama to /usr/local/bin...
>>> Creating ollama user...
>>> Adding ollama user to video group...
>>> Adding current user to ollama group...
>>> Creating ollama systemd service...
WARNING: Unable to detect NVIDIA/AMD GPU. Install lspci or lshw to automatically detect and install GPU dependencies.
>>> The Ollama API is now available at 127.0.0.1:11434.
>>> Install complete. Run "ollama" from the command line.
```

Next, we shall save the model to GGUF / llama.cpp

We clone `llama.cpp` and we default save it to `q8_0`. We allow all methods like `q4_k_m`. Use `save_pretrained_gguf` for local saving and `push_to_hub_gguf` for uploading to HF.

Some supported quant methods (full list on our Wiki page):

- `q8_0` - Fast conversion. High resource use, but generally acceptable.
- `q4_k_m` - Recommended. Uses Q6_K for half of the attention.wv and feed_forward.w2 tensors, else Q4_K.
- `q5_k_m` - Recommended. Uses Q6_K for half of the attention.wv and feed_forward.w2 tensors, else Q5_K.

We also support saving to multiple GGUF options in a list fashion! This can speed things up by 10 minutes or more if you want multiple export formats!

```
# Save to 8bit Q8_0
if True: model.save_pretrained_gguf("model", tokenizer,)
# Remember to go to https://huggingface.co/settings/tokens for a token!
# And change hf to your username!
if False: model.push_to_hub_gguf("hf/model", tokenizer, token = "")

# Save to 16bit GGUF
if False: model.save_pretrained_gguf("model", tokenizer, quantization_method = "f16")
if False: model.push_to_hub_gguf("hf/model", tokenizer, quantization_method = "f16", token = "")

# Save to q4_k_m GGUF
if False: model.save_pretrained_gguf("model", tokenizer, quantization_method = "q4_k_m")
if False: model.push_to_hub_gguf("hf/model", tokenizer, quantization_method = "q4_k_m", token = "")

# Save to multiple GGUF options - much faster if you want multiple!
if False:
    model.push_to_hub_gguf(
```

```
    "hf/model", # Change hf to your username!
    tokenizer,
    quantization_method = ["q4_k_m", "q8_0", "q5_k_m",],
    token = "", # Get a token at https://huggingface.co/settings/tokens
)
```

```
Unsloth: ##### The current model auto adds a BOS token.
Unsloth: ##### Your chat template has a BOS token. We shall remove it temporarily.
Unsloth: You have 1 CPUs. Using `safe_serialization` is 10x slower.
We shall switch to Pytorch saving, which will take 3 minutes and not 30 minutes.
To force `safe_serialization`, set it to `None` instead.
Unsloth: Kaggle/Colab has limited disk space. We need to delete the downloaded
model which will save 4-16GB of disk space, allowing you to save on Kaggle/Colab.
Unsloth: Will remove a cached repo with size 5.7G
Unsloth: Merging 4bit and LoRA weights to 16bit...
Unsloth: Will use up to 7.16 out of 12.67 RAM for saving.
 50%|████▌     | 16/32 [00:01<00:01, 10.44it/s]We will save to Disk and not RAM now.
100%|█████████| 32/32 [01:26<00:00,  2.70s/it]
Unsloth: Saving tokenizer... Done.
Unsloth: Saving model... This might take 5 minutes for Llama-7b...
Unsloth: Saving model/pytorch_model-00001-of-00004.bin...
Unsloth: Saving model/pytorch_model-00002-of-00004.bin...
Unsloth: Saving model/pytorch_model-00003-of-00004.bin...
Unsloth: Saving model/pytorch_model-00004-of-00004.bin...
Done.
Unsloth: Converting llama model. Can use fast conversion = False.
==((====))==  Unsloth: Conversion from QLoRA to GGUF information
   \\   /|    [0] Installing llama.cpp will take 3 minutes.
O^O/ \_/ \    [1] Converting HF to GUUF 16bits will take 3 minutes.
\        /    [2] Converting GGUF 16bits to ['q8_0'] will take 10 minutes each.
 "-____-"     In total, you will have to wait at least 16 minutes.

Unsloth: [0] Installing llama.cpp. This will take 3 minutes...
Unsloth: [1] Converting model at model into q8_0 GGUF format.
The output location will be ./model/unsloth.Q8_0.gguf
This will take 3 minutes...
INFO:hf-to-gguf:Loading model: model
INFO:gguf.gguf_writer:gguf: This GGUF file is for Little Endian only
INFO:hf-to-gguf:Set model parameters
INFO:hf-to-gguf:gguf: context length = 8192
INFO:hf-to-gguf:gguf: embedding length = 4096
INFO:hf-to-gguf:gguf: feed forward length = 14336
INFO:hf-to-gguf:gguf: head count = 32
INFO:hf-to-gguf:gguf: key-value head count = 8
INFO:hf-to-gguf:gguf: rope theta = 500000.0
INFO:hf-to-gguf:gguf: rms norm epsilon = 1e-05
INFO:hf-to-gguf:gguf: file type = 7
INFO:hf-to-gguf:Set model tokenizer
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
INFO:gguf.vocab:Adding 280147 merge(s).
INFO:gguf.vocab:Setting special token type bos to 128000
INFO:gguf.vocab:Setting special token type eos to 128001
INFO:gguf.vocab:Setting special token type pad to 128255
INFO:gguf.vocab:Setting chat_template to {{ 'Below describes some details about some passengers who went on the Titanic.
Predict whether they survived or perished based on their characteristics.
Output 1 if they survived, and 0 if they died.' }}{% for message in messages %}{% if message['role'] == 'user' %}{{ '
>>> Passenger Details:
' + message['content'] + '
' }}{% elif message['role'] == 'assistant' %}{{ '>>> Did they survive?
' + message['content'] + '<|end_of_text|>' }}{% else %}{{ raise_exception('Only user and assistant roles are supported!') }}{% endif %}{% endfor %}{% if ad
' }}{% endif %}
INFO:hf-to-gguf:Exporting model...
INFO:hf-to-gguf:gguf: loading model weight map from 'pytorch_model.bin.index.json'
INFO:hf-to-gguf:gguf: loading model part 'pytorch_model-00001-of-00004.bin'
```

We use `subprocess` to start `Ollama` up in a non blocking fashion! In your own desktop, you can simply open up a new `terminal` and type `ollama serve`, but in Colab, we have to use this hack!

```
import subprocess
subprocess.Popen(["ollama", "serve"])
import time
time.sleep(3) # Wait for a few seconds for Ollama to load!
```

`Ollama` needs a `Modelfile`, which specifies the model's prompt format. Let's print Unsloth's auto generated one:

```
print(tokenizer._ollama_modelfile)
```

```
FROM {__FILE_LOCATION__}

TEMPLATE """Below describes some details about some passengers who went on the Titanic.
Predict whether they survived or perished based on their characteristics.
Output 1 if they survived, and 0 if they died.{{ if .Prompt }}
>>> Passenger Details:
{{ .Prompt }}
{{ end }}>>> Did they survive?
{{ .Response }}<|end_of_text|>"""

PARAMETER stop "<|eot_id|>"
PARAMETER stop "<|start_header_id|>"
PARAMETER stop "<|end_header_id|>"
PARAMETER stop "<|end_of_text|>"
PARAMETER stop "<|reserved_special_token_"
```

We now will create an `Ollama` model called `unsloth_model` using the `Modelfile` which we auto generated!

```
!ollama create unsloth_model -f ./model/Modelfile
```