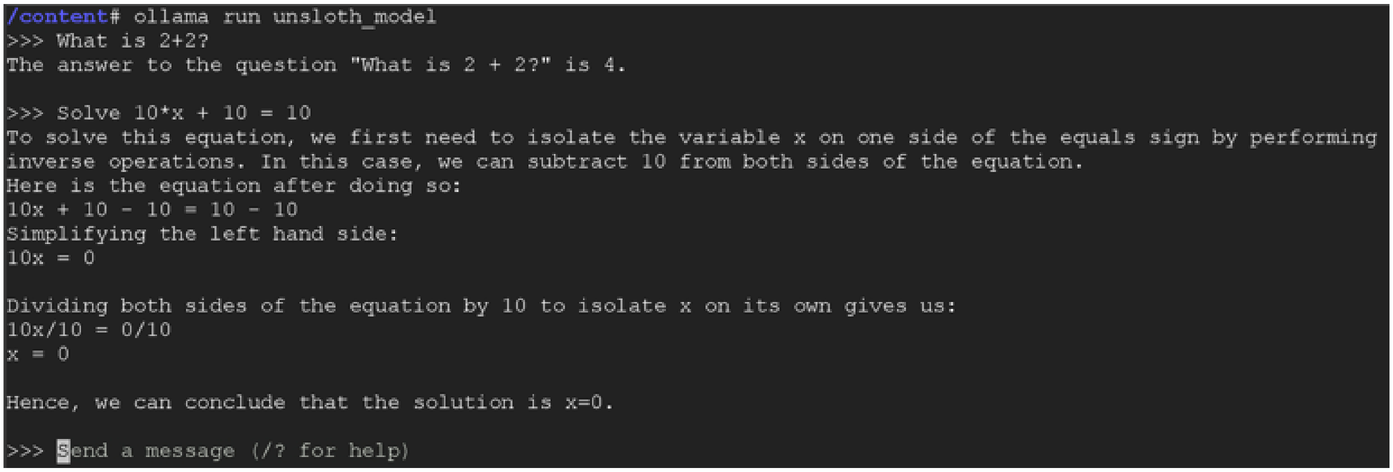
Tutorial: How to Finetune Llama-3 and Use In Ollama

Beginner's Guide for creating a customized personal assistant (like ChatGPT) to run locally on Ollama

By the end of this tutorial, you will create a custom chatbot by **finetuning Llama-3** with **Unsloth** for free. It can run locally via **Ollama** on your PC, or in a free GPU instance through **Google Colab**. You will be able to interact with the chatbot interactively like below:



```
/content# ollama run unsloth_model
>>> What is 2+2?
The answer to the question "What is 2 + 2?" is 4.

>>> Solve 10*x + 10 = 10
To solve this equation, we first need to isolate the variable x on one side of the equals sign by performing
inverse operations. In this case, we can subtract 10 from both sides of the equation.
Here is the equation after doing so:
10x + 10 - 10 = 10 - 10
Simplifying the left hand side:
10x = 0

Dividing both sides of the equation by 10 to isolate x on its own gives us:
10x/10 = 0/10
x = 0

Hence, we can conclude that the solution is x=0.

>>> Send a message (/? for help)
```

Unsloth makes finetuning much easier, and can automatically export the finetuned model to **Ollama** with integrated automatic `Modelfile` creation! If you need help, you can join our Discord server: https://discord.com/invite/unsloth

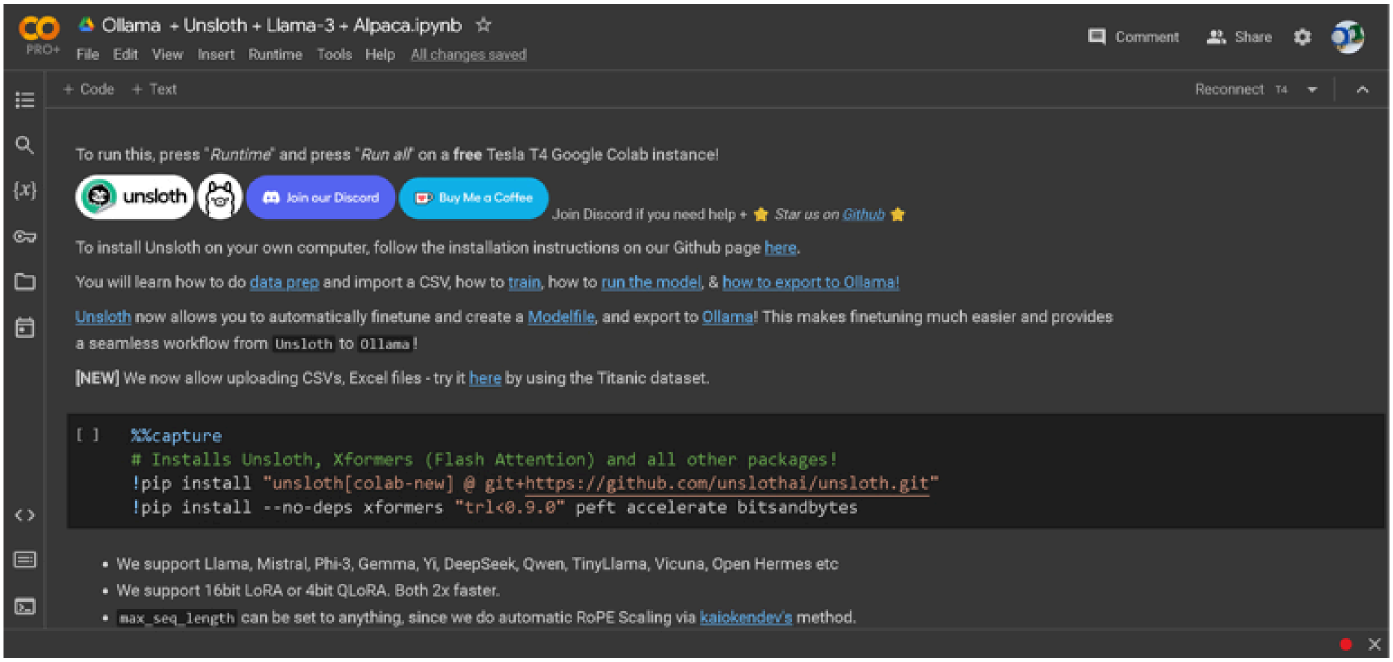
1. What is Unsloth?

Unsloth makes finetuning LLMs like Llama-3, Mistral, Phi-3 and Gemma 2x faster, use 70% less memory, and with no degradation in accuracy! We will be using Google Colab which provides a free GPU during this tutorial. You can access our free notebooks below:

- Ollama Llama-3 Alpaca (notebook which we will be using)

- CSV/Excel Ollama Guide

You will also need to login into your Google account!


Ollama + Unsloth + Llama-3 + Alpaca.ipynb
PRO+
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Reconnect T4
unsloth
Join our Discord
Buy Me a Coffee

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`!

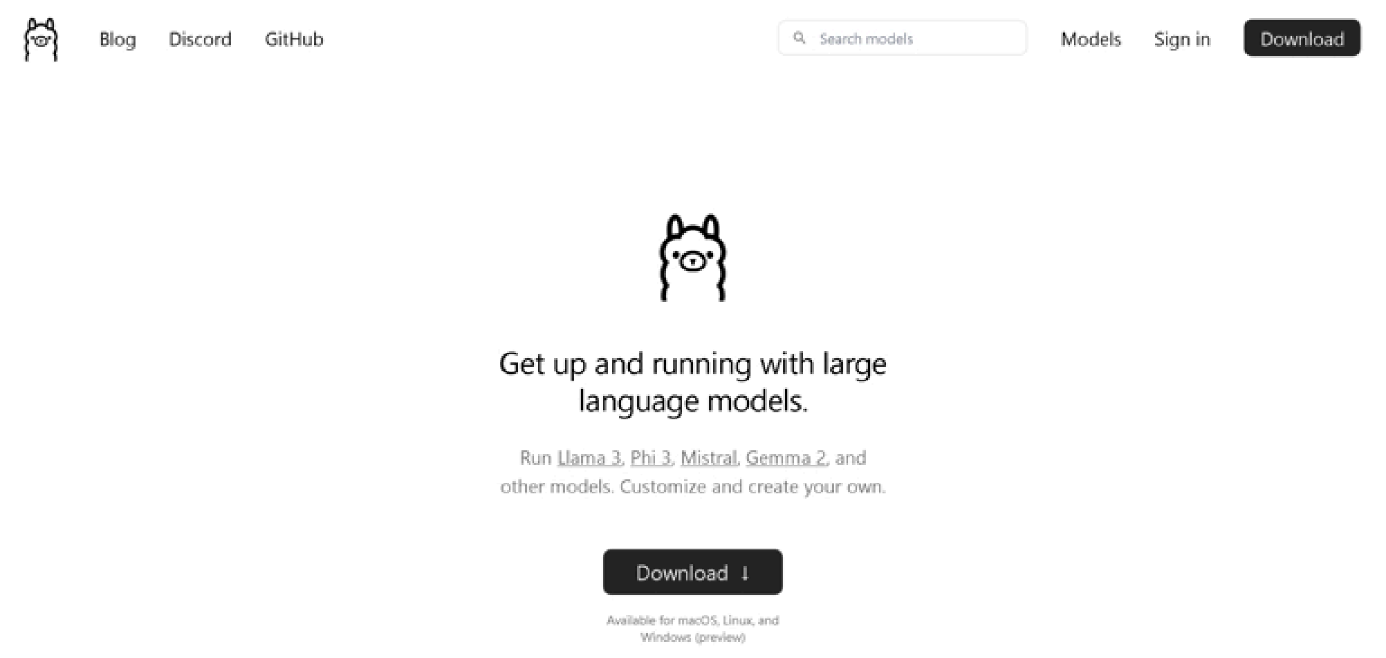
[NEW] We now allow uploading CSVs, Excel files - try it here by using the Titanic dataset.

```
[ ]  %%capture
     # Installs Unsloth, Xformers (Flash Attention) and all other packages!
     !pip install "unsloth[colab-new] @ git+https://github.com/unslothai/unsloth.git"
     !pip install --no-deps xformers "trl<0.9.0" peft accelerate bitsandbytes
```

- 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.

2. What is Ollama?

Ollama allows you to run language models from your own computer in a quick and simple way! It quietly launches a program which can run a language model like Llama-3 in the background. If you suddenly want to ask the language model a question, you can simply submit a request to Ollama, and it'll quickly return the results to you! We'll be using Ollama as our inference engine!

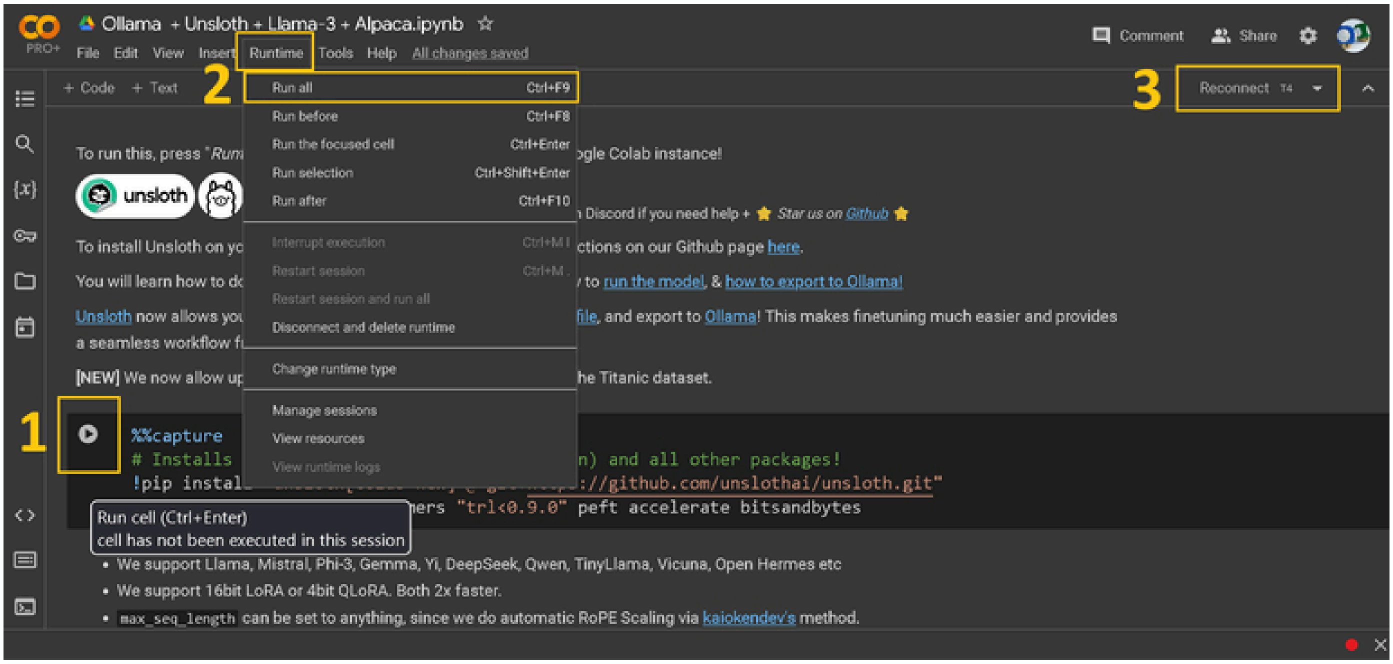


Get up and running with large language models.

Run Llama 3, Phi 3, Mistral, Gemma 2, and other models. Customize and create your own.

Available for macOS, Linux, and Windows (preview)

3. Install Unsloth



To run this, press "*Run* ogle Colab instance!

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

To install Unsloth on yo ctions on our Github page here.

You will learn how to do to run the model, & how to export to Ollama!

Unsloth now allows you file, and export to Ollama! This makes finetuning much easier and provides a seamless workflow f

[NEW] We now allow up he Titanic dataset.

```
%%capture
# Installs                      n) and all other packages!
!pip instal                      ://github.com/unslothai/unsloth.git"
                                 ers "trl<0.9.0" peft accelerate bitsandbytes
```

Run cell (Ctrl+Enter)
cell has not been executed in this session

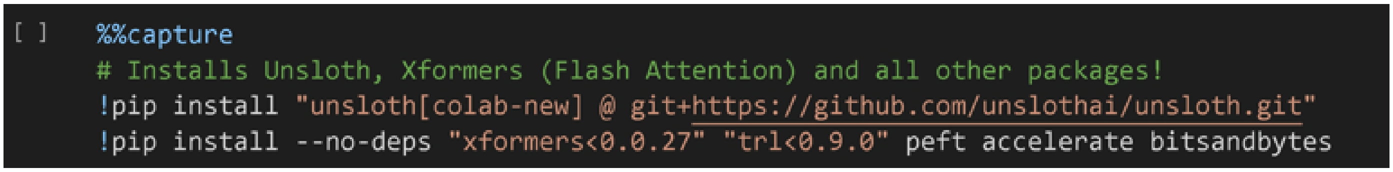
- We support Llama, Mistral, Phi-3, Gemma, Yi, DeepSeek, Qwen, TinyLlama, Vicuna, Open Hermes etc
- We support 16bit LoRA or 4bit QLoRA. Both 2x faster.
- `max_seq_length` can be set to anything, since we do automatic RoPE Scaling via kaiokendev's method.

If you have never used a Colab notebook, a quick primer on the notebook itself:

1. **Play Button at each "cell".** Click on this to run that cell's code. You must not skip any cells and you must run every cell in chronological order. If you encounter any errors, simply rerun the cell you did not run before. Another option is to click CTRL + ENTER if you don't want to click the play button.

2. **Runtime Button in the top toolbar.** You can also use this button and hit "Run all" to run the entire notebook in 1 go. This will skip all the customization steps, and can be a good first try.
3. **Connect / Reconnect T4 button.** You can click here for more advanced system statistics.

The first installation cell looks like below: Remember to click the PLAY button in the brackets [ ]. We grab our open source Github package, and install some other packages.

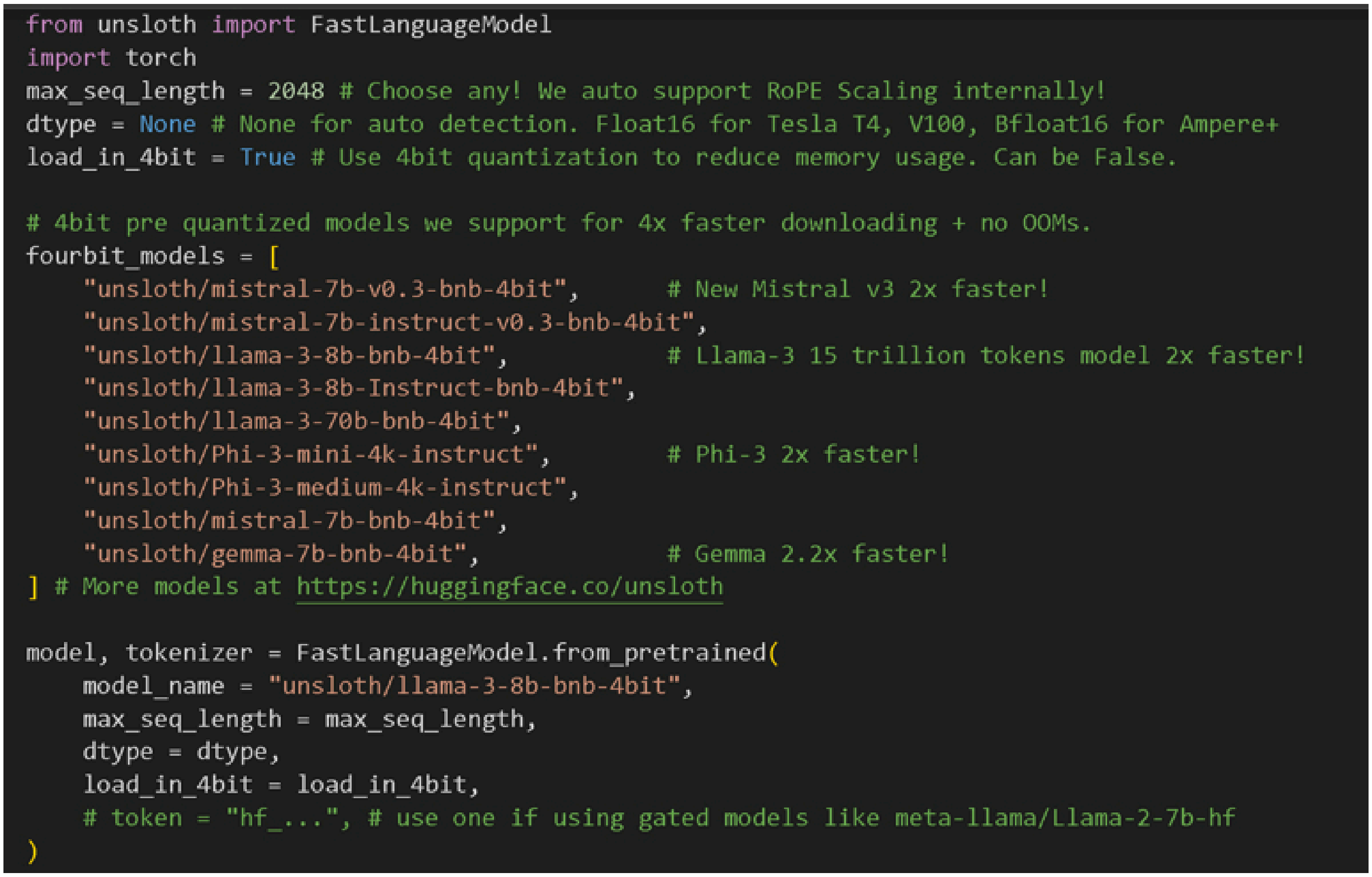


```
[ ]   %%capture
      # Installs Unsloth, Xformers (Flash Attention) and all other packages!
      !pip install "unsloth[colab-new] @ git+https://github.com/unslothai/unsloth.git"
      !pip install --no-deps "xformers<0.0.27" "trl<0.9.0" peft accelerate bitsandbytes
```

4. Selecting a model to finetune

Let's now select a model for finetuning! We defaulted to Llama-3 from Meta / Facebook which was trained on a whopping 15 trillion "tokens". Assume a token is like 1 English word. That's approximately 350,000 thick Encyclopedias worth! Other popular models include Mistral, Phi-3 (trained using GPT-4 output) and Gemma from Google (13 trillion tokens!).

Unsloth supports these models and more! In fact, simply type a model from the Hugging Face model hub to see if it works! We'll error out if it doesn't work.



```
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
)
```

There are 3 other settings which you can toggle:

1.
   ```
   max_seq_length = 2048
   ```

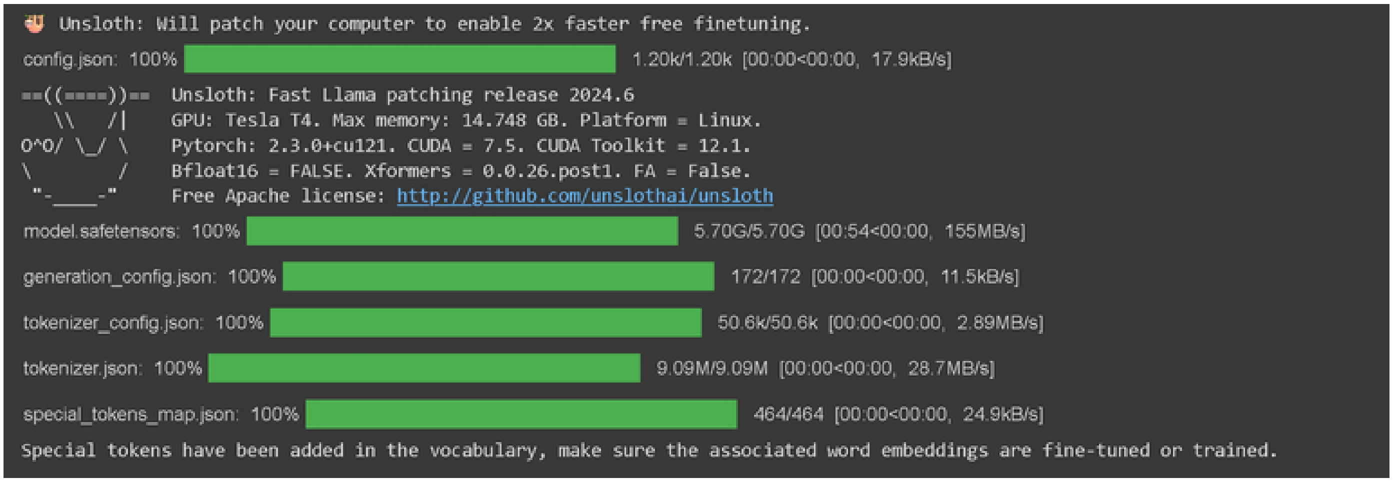
   This determines the context length of the model. Gemini for example has over 1 million context length, whilst Llama-3 has 8192 context length. We allow you to select ANY number - but we recommend setting it 2048 for testing purposes. Unsloth also supports very long context finetuning, and we show we can provide 4x longer context lengths than the best.

2.
   ```
   dtype = None
   ```

   Keep this as None, but you can select torch.float16 or torch.bfloat16 for newer GPUs.

3.
   ```
   load_in_4bit = True
   ```

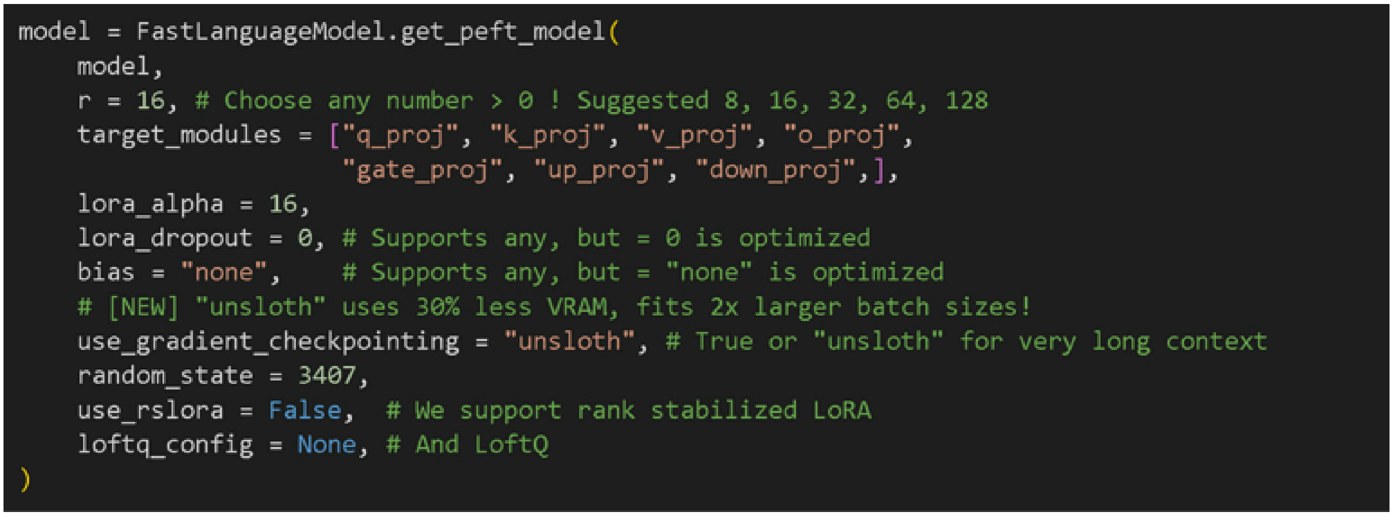
   We do finetuning in 4 bit quantization. This reduces memory usage by 4x, allowing us to actually do finetuning in a free 16GB memory GPU. 4 bit quantization essentially converts weights into a limited set of numbers to reduce memory usage. A drawback of this is there is a 1-2% accuracy degradation. Set this to False on larger GPUs like H100s if you want that tiny extra accuracy.



```
🦥 Unsloth: Will patch your computer to enable 2x faster free finetuning.
config.json: 100% 1.20k/1.20k [00:00<00:00, 17.9kB/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 [00:54<00:00, 155MB/s]
generation_config.json: 100% 172/172 [00:00<00:00, 11.5kB/s]
tokenizer_config.json: 100% 50.6k/50.6k [00:00<00:00, 2.89MB/s]
tokenizer.json: 100% 9.09M/9.09M [00:00<00:00, 28.7MB/s]
special_tokens_map.json: 100% 464/464 [00:00<00:00, 24.9kB/s]
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
```

If you run the cell, you will get some print outs of the Unsloth version, which model you are using, how much memory your GPU has, and some other statistics. Ignore this for now.

5. Parameters for finetuning



```
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
)
```

Now to customize your finetune, you can edit the numbers above, but you can ignore it, since we already select quite reasonable numbers.

The goal is to change these numbers to increase accuracy, but also **counteract over-fitting**. Over-fitting is when you make the language model memorize a dataset, and not be able to answer novel new questions. We want to a final model to answer unseen questions, and not do memorization.

1.
   ```
   r = 16, # Choose any number > 0 ! Suggested 8, 16, 32, 64, 128
   ```

The rank of the finetuning process. A larger number uses more memory and will be slower, but can increase accuracy on harder tasks. We normally suggest numbers like 8 (for fast finetunes), and up to 128. Too large numbers can causing over-fitting, damaging your model's quality.

2.
   ```
   target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                     "gate_proj", "up_proj", "down_proj",],
   ```

   We select all modules to finetune. You can remove some to reduce memory usage and make training faster, but we highly do not suggest this. Just train on all modules!

3.
   ```
   lora_alpha = 16,
   ```

   The scaling factor for finetuning. A larger number will make the finetune learn more about your dataset, but can promote over-fitting. We suggest this to equal to the rank `r`, or double it.

4.
   ```
   lora_dropout = 0, # Supports any, but = 0 is optimized
   ```

   Leave this as 0 for faster training! Can reduce over-fitting, but not that much.

5.
   ```
   bias = "none",    # Supports any, but = "none" is optimized
   ```

   Leave this as 0 for faster and less over-fit training!

6.
   ```
   use_gradient_checkpointing = "unsloth", # True or "unsloth" for very long con
   ```

   Options include `True`, `False` and `"unsloth"`. We suggest `"unsloth"` since we reduce memory usage by an extra 30% and support extremely long context finetunes.You can read up here: https://unsloth.ai/blog/long-context for more details.

7.
   ```
   random_state = 3407,
   ```

   The number to determine deterministic runs. Training and finetuning needs random numbers, so setting this number makes experiments reproducible.

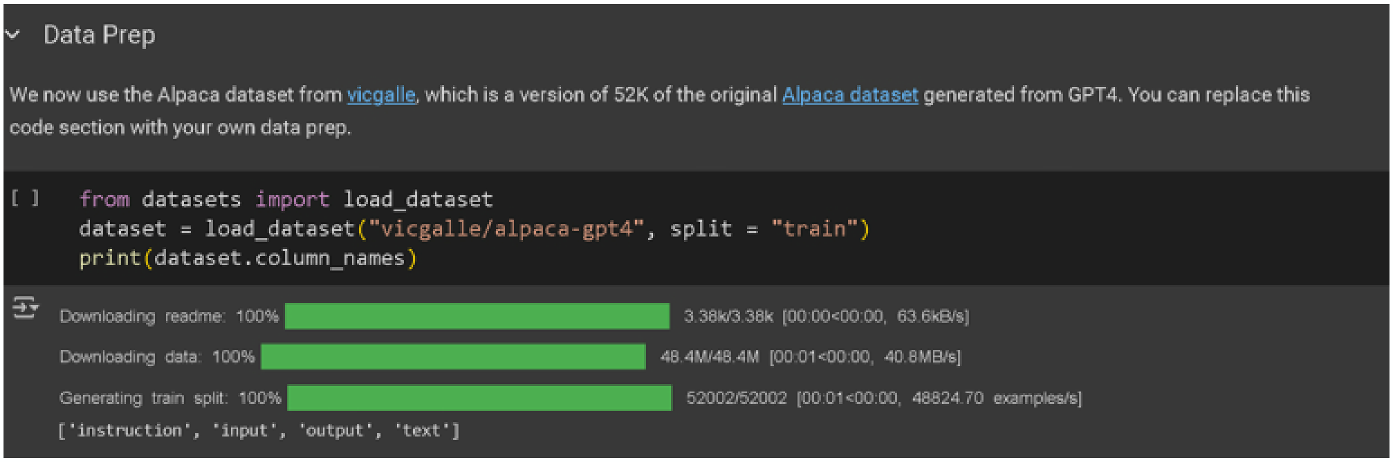
8.
   ```
   use_rslora = False,  # We support rank stabilized LoRA
   ```

   Advanced feature to set the `lora_alpha = 16` automatically. You can use this if you want!

9.
   ```
   loftq_config = None, # And LoftQ
   ```

   Advanced feature to initialize the LoRA matrices to the top r singular vectors of the weights. Can improve accuracy somewhat, but can make memory usage explode at the start.

6. Alpaca Dataset



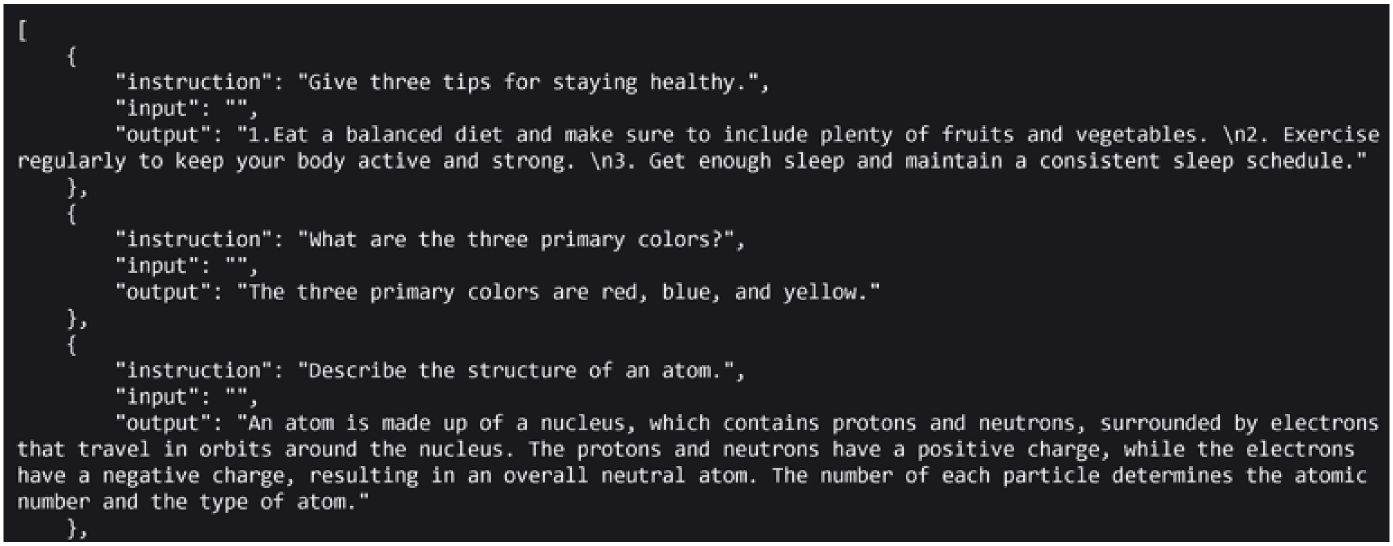
We now use the Alpaca dataset from vicgalle, which is a version of 52K of the original Alpaca dataset generated from GPT4. You can replace this code section with your own data prep.

```
[ ]  from datasets import load_dataset
     dataset = load_dataset("vicgalle/alpaca-gpt4", split = "train")
     print(dataset.column_names)
```

```
Downloading readme: 100%  3.38k/3.38k [00:00<00:00, 63.6kB/s]
Downloading data: 100%  48.4M/48.4M [00:01<00:00, 40.8MB/s]
Generating train split: 100%  52002/52002 [00:01<00:00, 48824.70 examples/s]
['instruction', 'input', 'output', 'text']
```

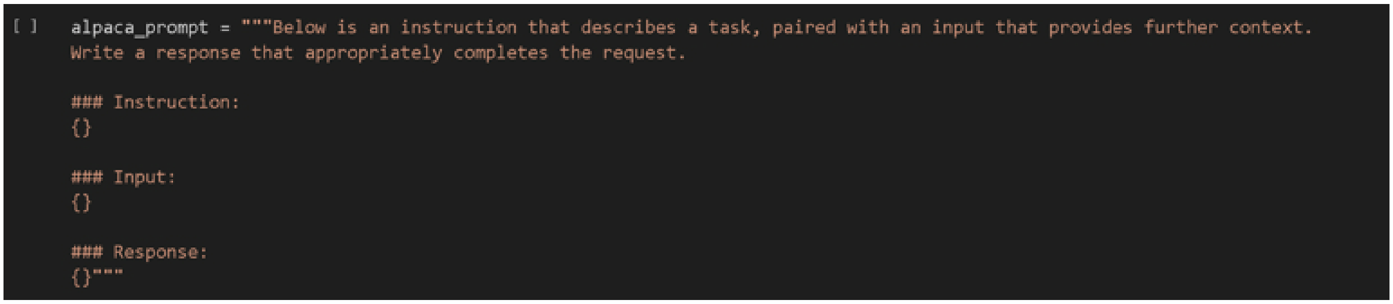
We will now use the Alpaca Dataset created by calling GPT-4 itself. It is a list of 52,000 instructions and outputs which was very popular when Llama-1 was released, since it made finetuning a base LLM be competitive with ChatGPT itself.

You can access the GPT4 version of the Alpaca dataset here: https://huggingface.co/datasets/vicgalle/alpaca-gpt4. An older first version of the dataset is here: https://github.com/tatsu-lab/stanford_alpaca. Below shows some examples of the dataset:



```
[
    {
        "instruction": "Give three tips for staying healthy.",
        "input": "",
        "output": "1.Eat a balanced diet and make sure to include plenty of fruits and vegetables. \n2. Exercise
regularly to keep your body active and strong. \n3. Get enough sleep and maintain a consistent sleep schedule."
    },
    {
        "instruction": "What are the three primary colors?",
        "input": "",
        "output": "The three primary colors are red, blue, and yellow."
    },
    {
        "instruction": "Describe the structure of an atom.",
        "input": "",
        "output": "An atom is made up of a nucleus, which contains protons and neutrons, surrounded by electrons
that travel in orbits around the nucleus. The protons and neutrons have a positive charge, while the electrons
have a negative charge, resulting in an overall neutral atom. The number of each particle determines the atomic
number and the type of atom."
    },
```

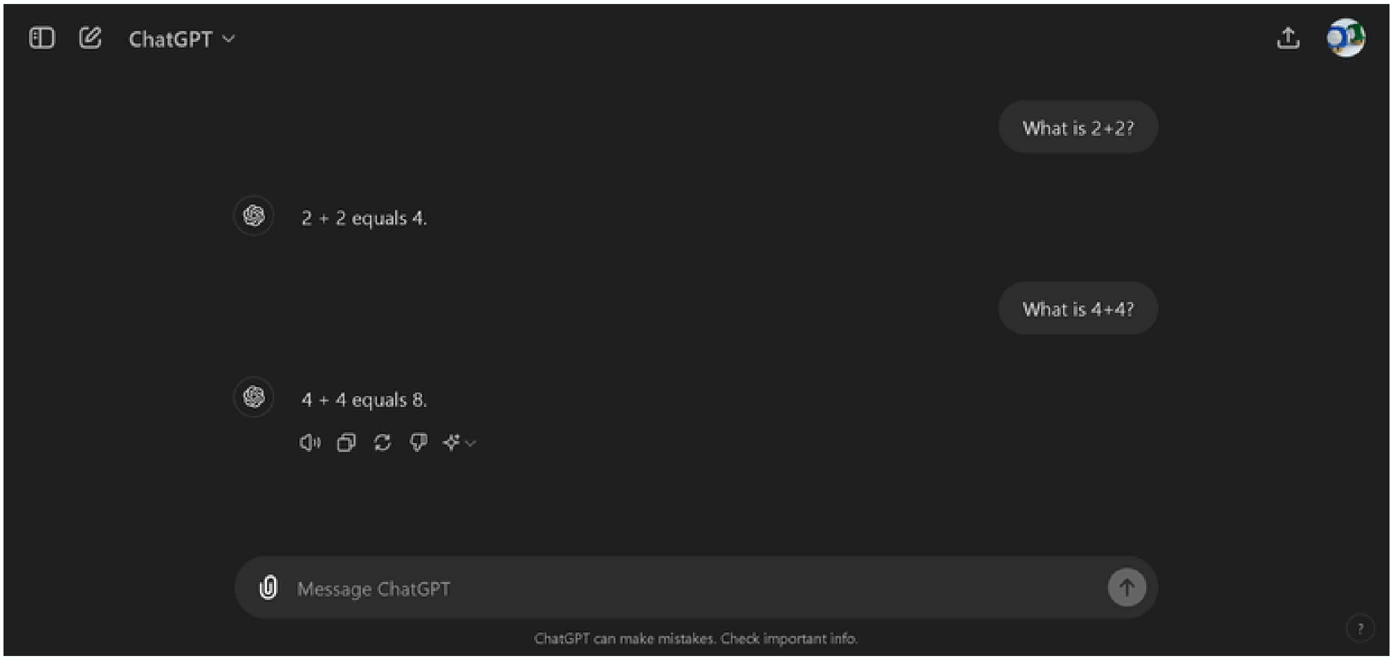
You can see there are 3 columns in each row - an instruction, and input and an output. We essentially combine each row into 1 large prompt like below. We then use this to finetune the language model, and this made it very similar to ChatGPT. We call this process **supervised instruction finetuning**.



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

### Instruction:
{}

### Input:
{}

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

7. Multiple columns for finetuning

But a big issue is for ChatGPT style assistants, we only allow 1 instruction / 1 prompt, and not multiple columns / inputs. For example in ChatGPT, you can see we must submit 1 prompt, and not multiple prompts.



What is 2+2?

2 + 2 equals 4.

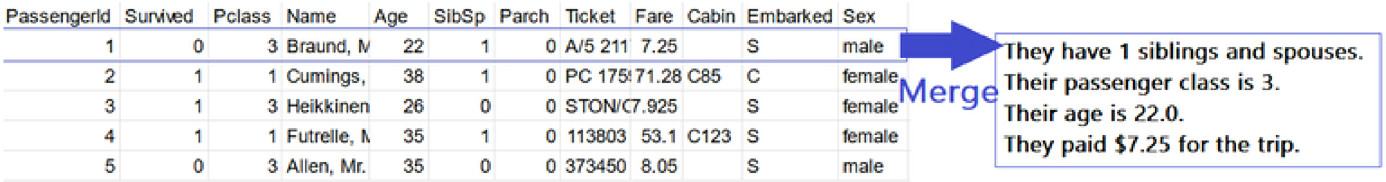
What is 4+4?

4 + 4 equals 8.

ChatGPT can make mistakes. Check important info.

This essentially means we have to "merge" multiple columns into 1 large prompt for finetuning to actually function!

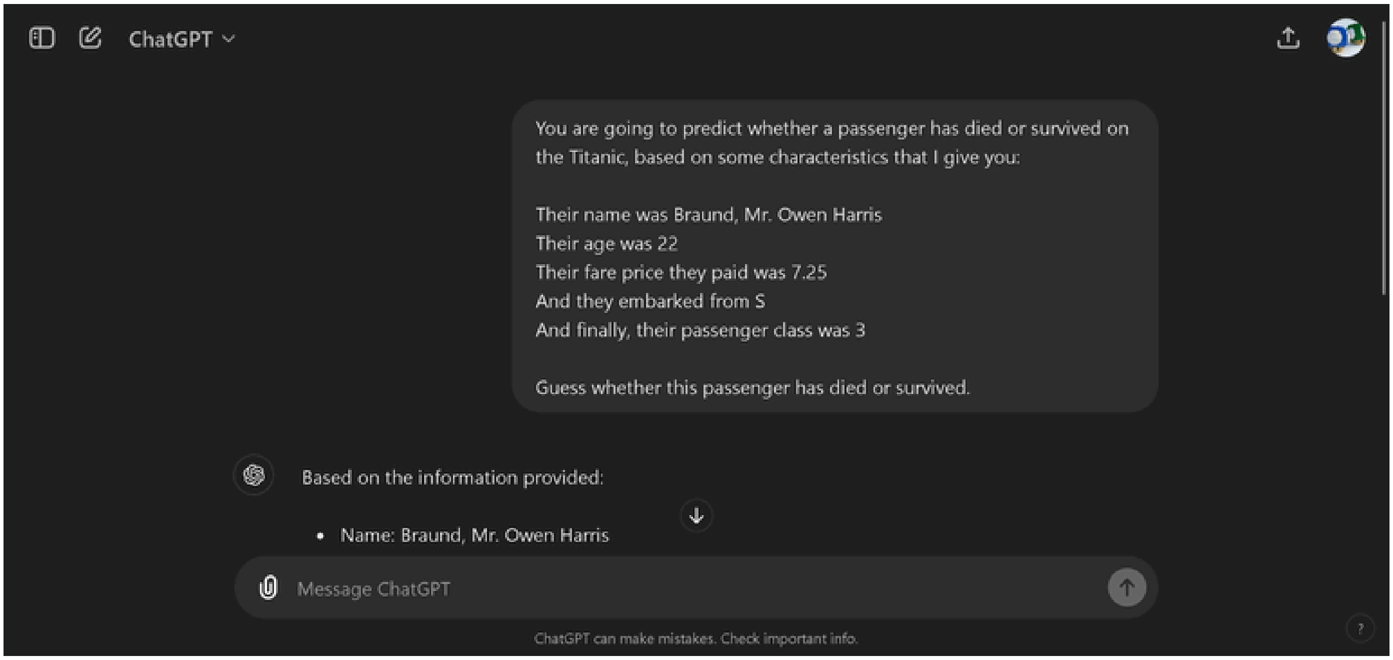
For example the very famous Titanic dataset has many many columns. Your job was to predict whether a passenger has survived or died based on their age, passenger class, fare price etc. We can't simply pass this into ChatGPT, but rather, we have to "merge" this information into 1 large prompt.



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

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

For example, if we ask ChatGPT with our "merged" single prompt which includes all the information for that passenger, we can then ask it to guess or predict whether the passenger has died or survived.



You are going to predict whether a passenger has died or survived on the Titanic, based on some characteristics that I give you:

Their name was Braund, Mr. Owen Harris
Their age was 22
Their fare price they paid was 7.25
And they embarked from S
And finally, their passenger class was 3

Guess whether this passenger has died or survived.

Based on the information provided:

- Name: Braund, Mr. Owen Harris

ChatGPT can make mistakes. Check important info.

Other finetuning libraries require you to manually prepare your dataset for finetuning, by merging all your columns into 1 prompt. In Unsloth, we simply provide the function called `to_sharegpt` which does this in 1 go!

To access the Titanic finetuning notebook or if you want to upload a CSV or Excel file, go here: https://colab.research.google.com/drive/1VYkncZMfGFkeCEgN2IzbZIKEDkyQuJAS?usp=sharing

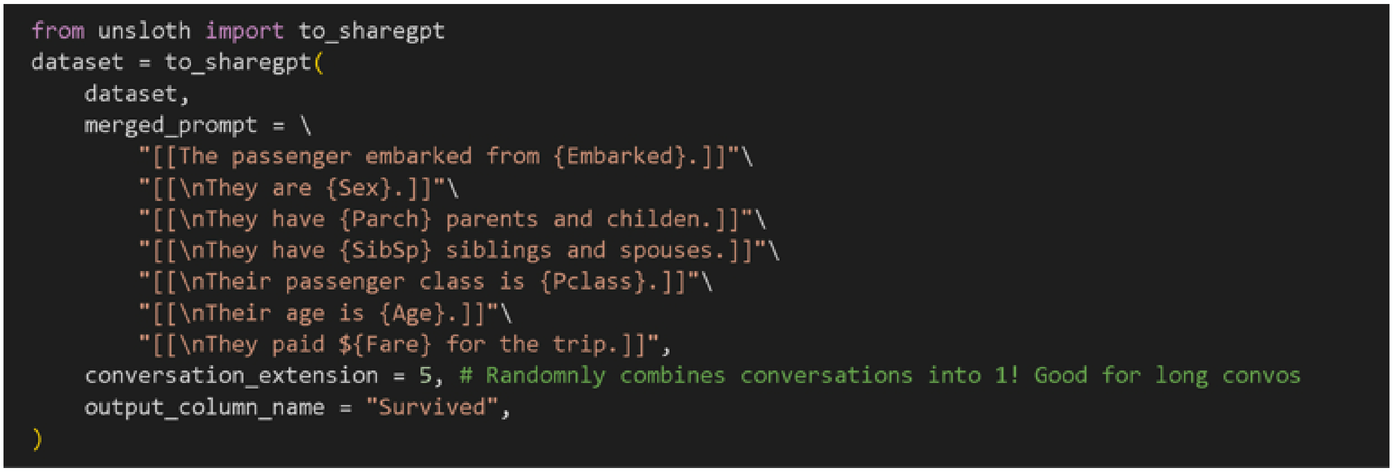
```python
from unsloth import to_sharegpt
dataset = to_sharegpt(
    dataset,
    merged_prompt = "{instruction}[[\nYour input is:\n{input}]]",
    output_column_name = "output",
    conversation_extension = 3, # Select more to handle longer conversations
)
```

```
Merging columns: 100% 52002/52002 [00:00<00:00, 83429.16 examples/s]
Converting to ShareGPT: 100% 52002/52002 [00:01<00:00, 57482.52 examples/s]
```

Now this is a bit more complicated, since we allow a lot of customization, but there are a few points:

- You must enclose all columns in curly braces `{}`. These are the column names in the actual CSV / Excel file.
- Optional text components must be enclosed in `[[]]`. For example if the column "input" is empty, the merging function will not show the text and skip this. This is useful for datasets with missing values.
- Select the output or target / prediction column in `output_column_name`. For the Alpaca dataset, this will be `output`.

For example in the Titanic dataset, we can create a large merged prompt format like below, where each column / piece of text becomes optional.



```
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",
)
```

For example, pretend the dataset looks like this with a lot of missing data:

Embarked	Age	Fare
S	23	
	18	7.25

Then, we do not want the result to be:

1. The passenger embarked from S. Their age is 23. Their fare is **EMPTY**.
2. The passenger embarked from **EMPTY**. Their age is 18. Their fare is $7.25.

Instead by optionally enclosing columns using `[[]]`, we can exclude this information entirely.

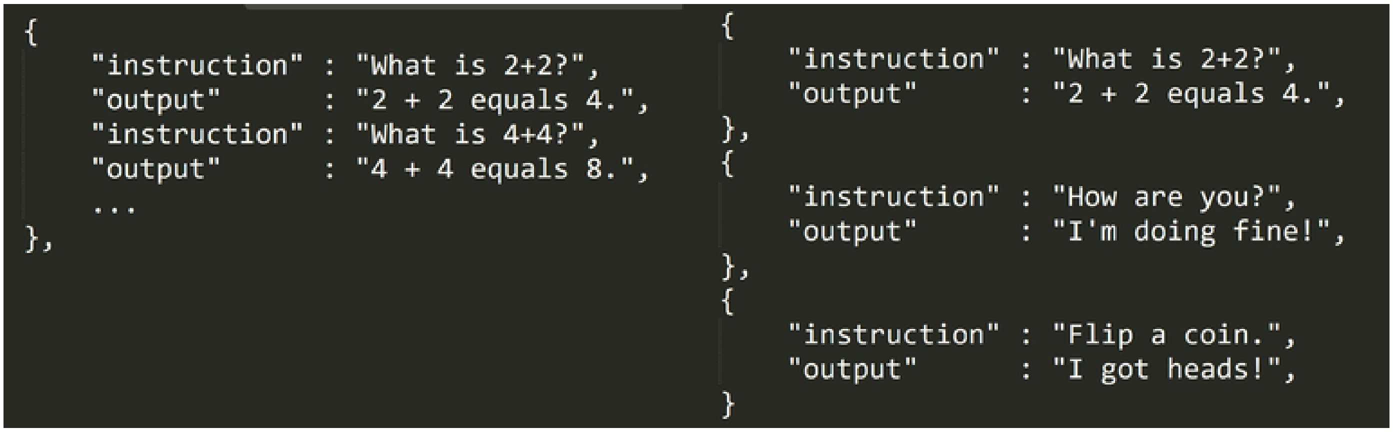
1. [[The passenger embarked from S.]] [[Their age is 23.]] [[Their fare is **EMPTY**.]]
2. [[The passenger embarked from **EMPTY**.]] [[Their age is 18.]] [[Their fare is $7.25.]]

becomes:

1. The passenger embarked from S. Their age is 23.
2. Their age is 18. Their fare is $7.25.

8. Multi turn conversations

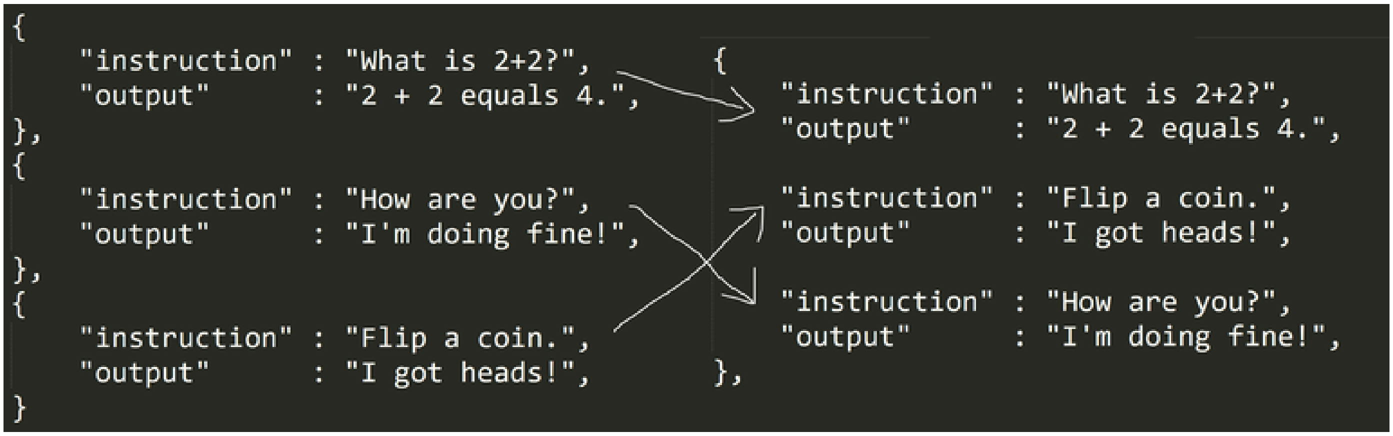
A bit issue if you didn't notice is the Alpaca dataset is single turn, whilst remember using ChatGPT was interactive and you can talk to it in multiple turns. For example, the left is what we want, but the right which is the Alpaca dataset only provides singular conversations. We want the finetuned language model to somehow learn how to do multi turn conversations just like ChatGPT.



```
{
    "instruction" : "What is 2+2?",
    "output"      : "2 + 2 equals 4.",
    "instruction" : "What is 4+4?",
    "output"      : "4 + 4 equals 8.",
    ...
},
```

```
{
    "instruction" : "What is 2+2?",
    "output"      : "2 + 2 equals 4.",
},
{
    "instruction" : "How are you?",
    "output"      : "I'm doing fine!",
},
{
    "instruction" : "Flip a coin.",
    "output"      : "I got heads!",
}
```

So we introduced the `conversation_extension` parameter, which essentially selects some random rows in your single turn dataset, and merges them into 1 conversation! For example, if you set it to 3, we randomly select 3 rows and merge them into 1! Setting them too long can make training slower, but could make your chatbot and final finetune much better!

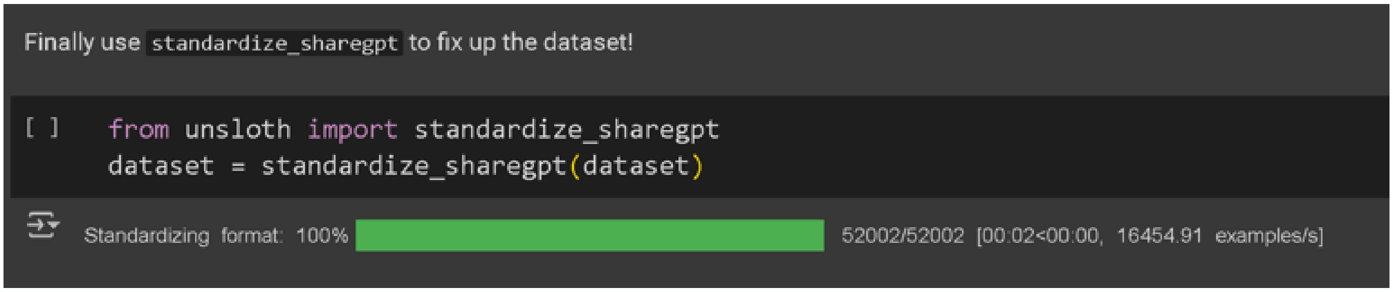


```
{
    "instruction" : "What is 2+2?",
    "output"      : "2 + 2 equals 4.",
},
{
    "instruction" : "How are you?",
    "output"      : "I'm doing fine!",
},
{
    "instruction" : "Flip a coin.",
    "output"      : "I got heads!",
}
```

```
{
    "instruction" : "What is 2+2?",
    "output"      : "2 + 2 equals 4.",

    "instruction" : "Flip a coin.",
    "output"      : "I got heads!",

    "instruction" : "How are you?",
    "output"      : "I'm doing fine!",
},
```

Then set `output_column_name` to the prediction / output column. For the Alpaca dataset dataset, it would be the output column.

We then use the `standardize_sharegpt` function to just make the dataset in a correct format for finetuning! Always call this!



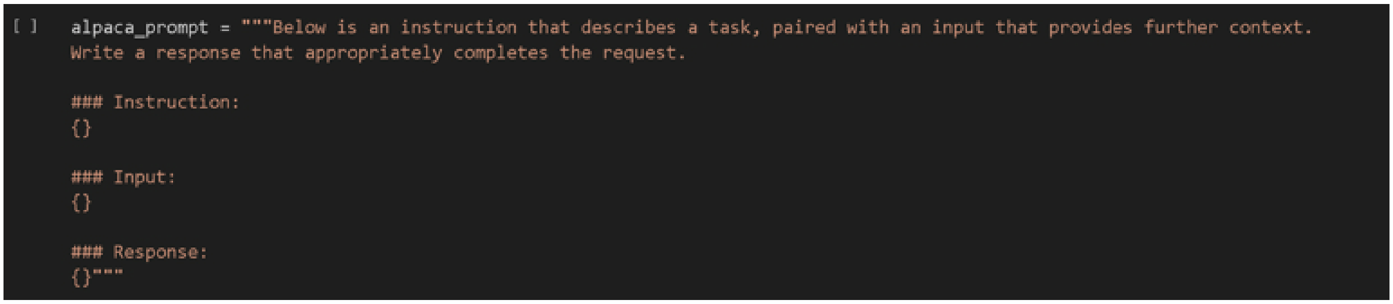
Finally use `standardize_sharegpt` to fix up the dataset!

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

```
Standardizing format: 100% 52002/52002 [00:02<00:00, 16454.91 examples/s]
```

9. Customizable Chat Templates

We can now specify the chat template for finetuning itself. The very famous Alpaca format is below:

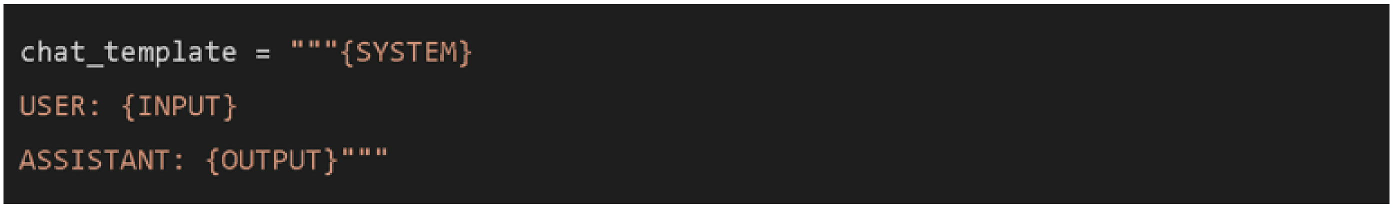


```
[ ]  alpaca_prompt = """Below is an instruction that describes a task, paired with an input that provides further context.
     Write a response that appropriately completes the request.

     ### Instruction:
     {}

     ### Input:
     {}

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

But remember we said this was a bad idea because ChatGPT style finetunes require only 1 prompt? Since we successfully merged all dataset columns into 1 using Unsloth, we essentially can create the below style chat template with 1 input column (instruction) and 1 output:

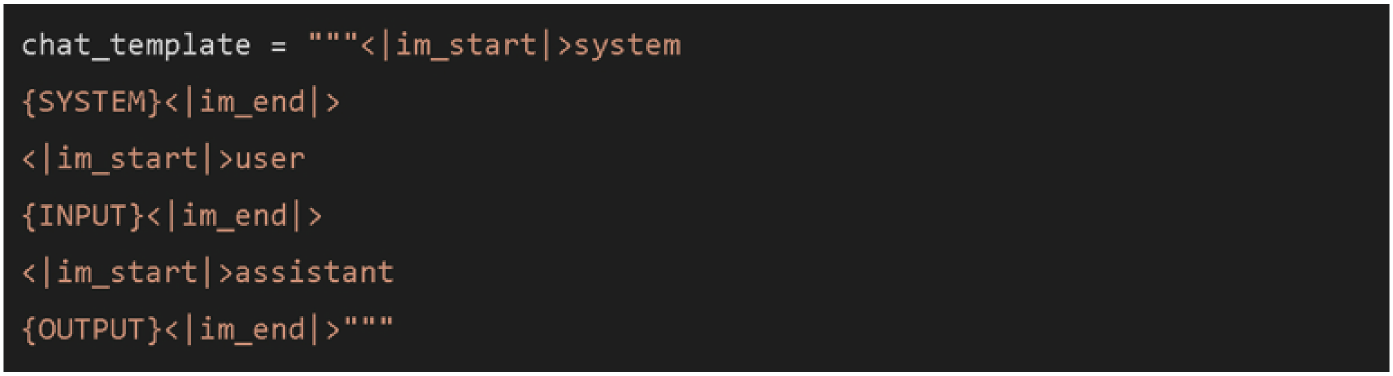
```
chat_template = """Below are some instructions that describe some tasks. Write
responses that appropriately complete each request.

### Instruction:
{INPUT}

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

We just require you must put a `{INPUT}` field for the instruction and an `{OUTPUT}` field for the model's output field. We in fact allow an optional `{SYSTEM}` field as well which is useful to customize a system prompt just like in ChatGPT. For example, below are some cool examples which you can customize the chat template to be:



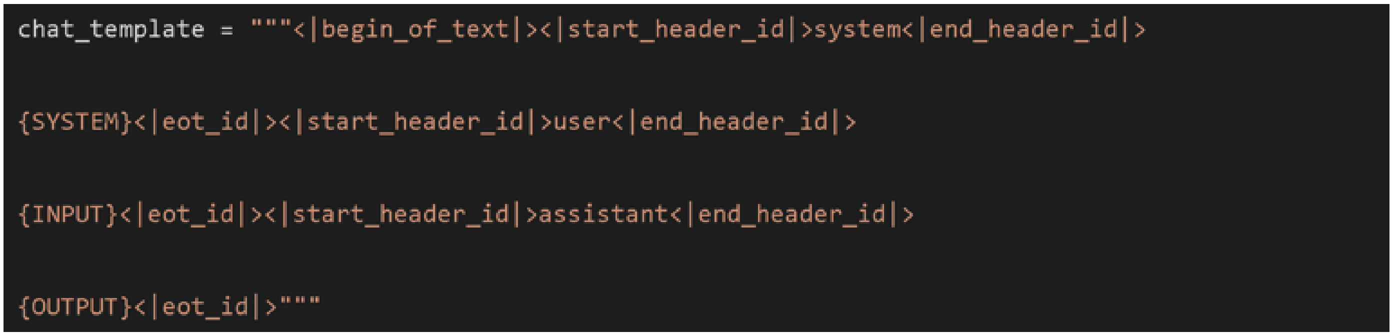
```
chat_template = """{SYSTEM}
USER: {INPUT}
ASSISTANT: {OUTPUT}"""
```

For the ChatML format used in OpenAI models:



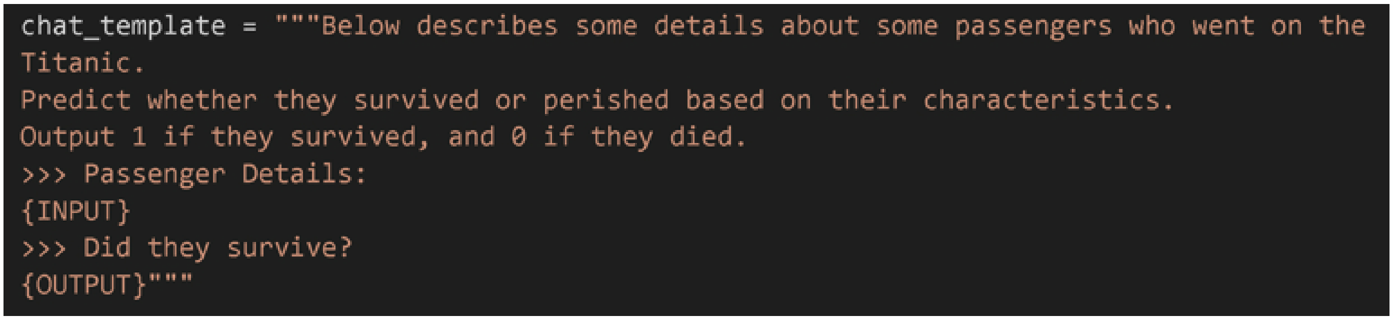
```
chat_template = """<|im_start|>system
{SYSTEM}<|im_end|>
<|im_start|>user
{INPUT}<|im_end|>
<|im_start|>assistant
{OUTPUT}<|im_end|>"""
```

Or you can use the Llama-3 template itself (which only functions by using the instruct version of Llama-3): We in fact allow an optional `{SYSTEM}` field as well which is useful to customize a system prompt just like in ChatGPT.



```
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|>"""
```

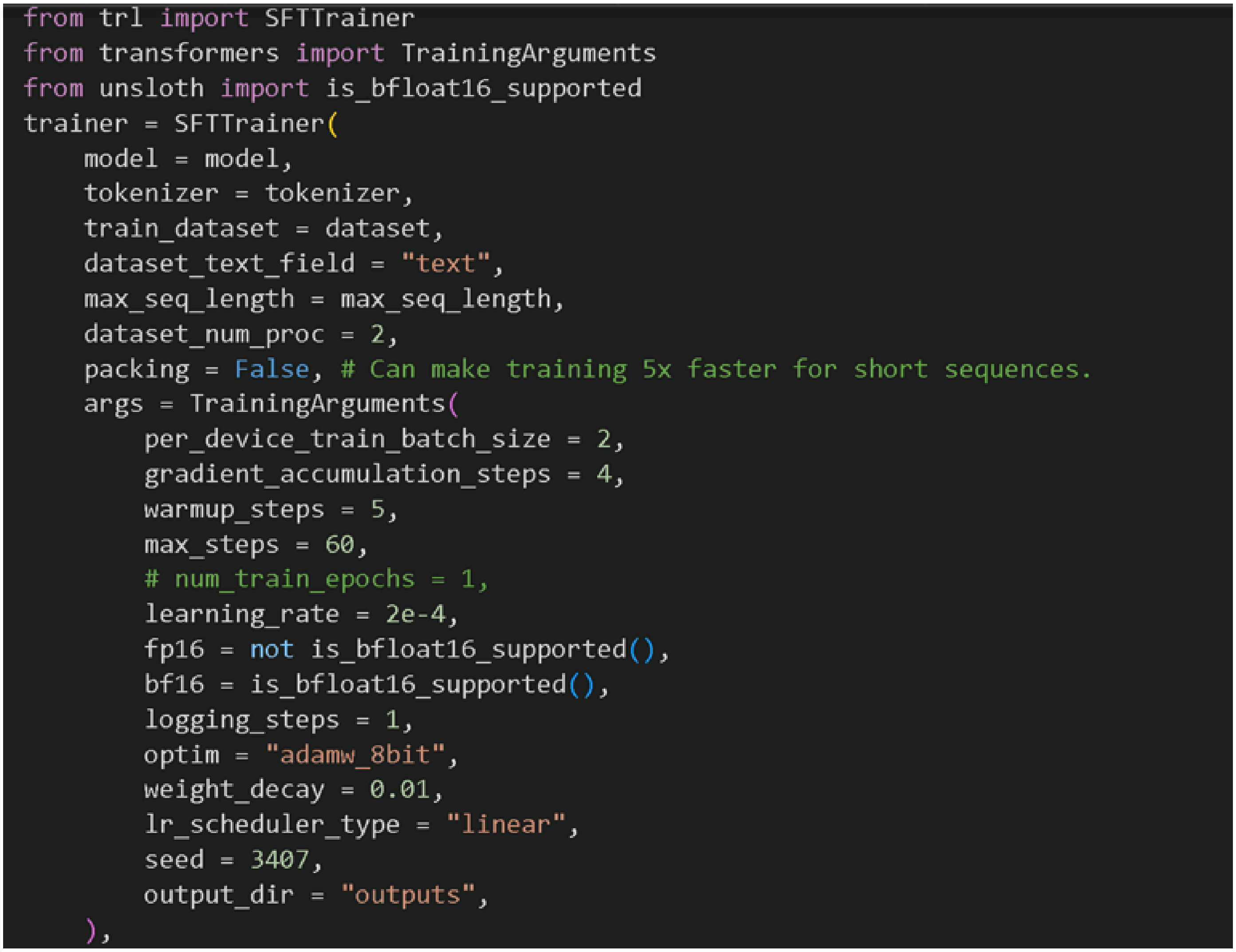
Or in the Titanic prediction task where you had to predict if a passenger died or survived in this Colab notebook which includes CSV and Excel uploading: https://colab.research.google.com/drive/1VYkncZMfGFkeCEgN2IzbZIKEDkyQuJAS?usp=sharing



```
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}"""
```

10. Train the model

Let's train the model now! We normally suggest people to not edit the below, unless if you want to finetune for longer steps or want to train on large batch sizes.



```
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,
        # num_train_epochs = 1,
        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",
    ),
```

We do not normally suggest changing the parameters above, but to elaborate on some of them:

1.
   ```
   per_device_train_batch_size = 2,
   ```

   Increase the batch size if you want to utilize the memory of your GPU more. Also increase this to make training more smooth and make the process not over-fit. We normally do not suggest this, since this might make training actually slower due to padding issues. We normally instead ask you to increase `gradient_accumulation_steps` which just does more passes over the dataset.

2.
   ```
   gradient_accumulation_steps = 4,
   ```

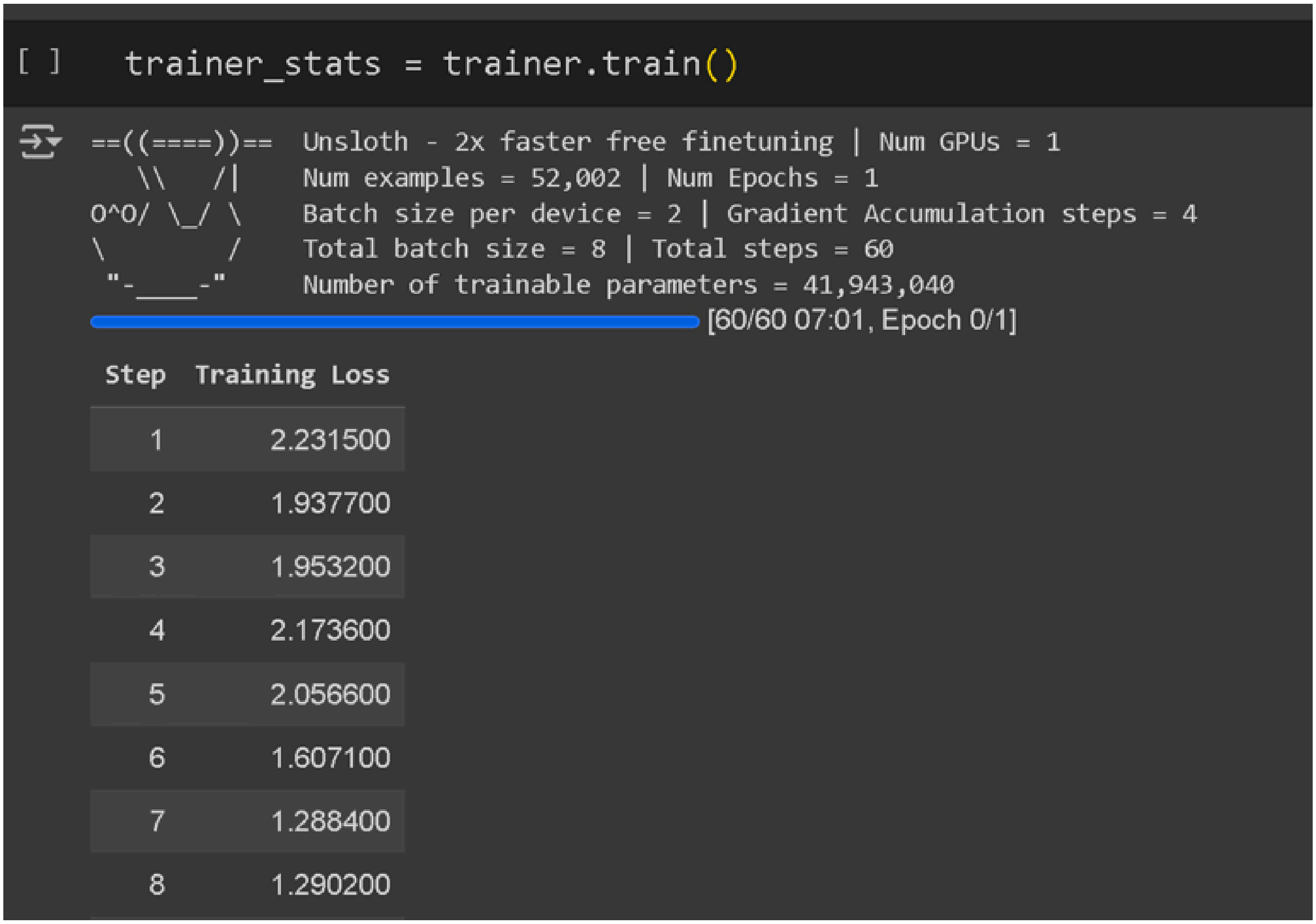
   Equivalent to increasing the batch size above itself, but does not impact memory consumption! We normally suggest people increasing this if you want smoother training loss curves.

3.
   ```
   max_steps = 60, # num_train_epochs = 1,
   ```

We set steps to 60 for faster training. For full training runs which can take hours, instead comment out `max_steps`, and replace it with `num_train_epochs = 1`. Setting it to 1 means 1 full pass over your dataset. We normally suggest 1 to 3 passes, and no more, otherwise you will over-fit your finetune.

4.
```
learning_rate = 2e-4,
```

   Reduce the learning rate if you want to make the finetuning process slower, but also converge to a higher accuracy result most likely. We normally suggest 2e-4, 1e-4, 5e-5, 2e-5 as numbers to try.



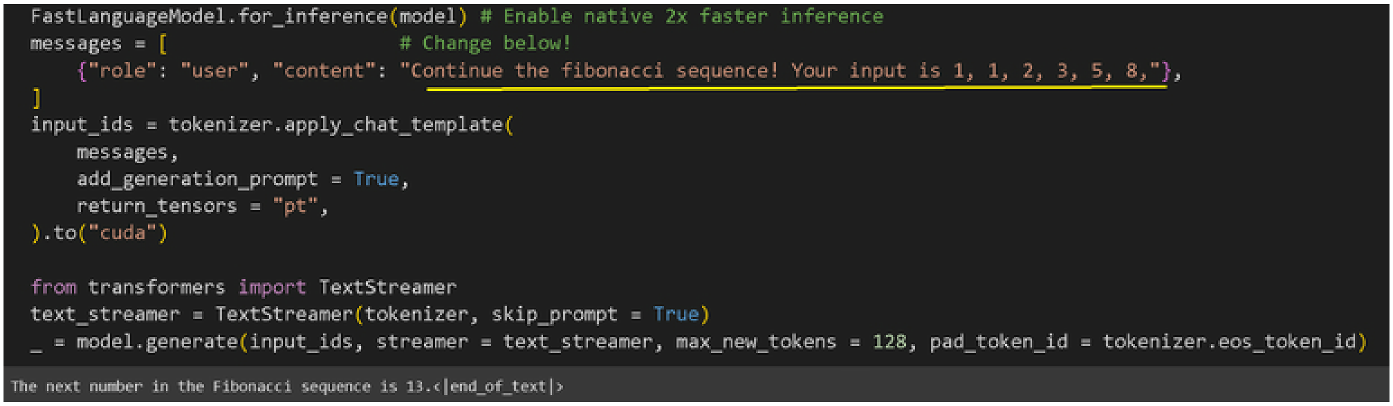
```
[ ]  trainer_stats = trainer.train()
```

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs = 1
   \\   /|    Num examples = 52,002 | 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 07:01, Epoch 0/1]
```

Step	Training Loss
1	2.231500
2	1.937700
3	1.953200
4	2.173600
5	2.056600
6	1.607100
7	1.288400
8	1.290200

You will see a log of some numbers! This is the training loss, and your job is to set parameters to make this go to as close to 0.5 as possible! If your finetune is not reaching 1, 0.8 or 0.5, you might have to adjust some numbers. If your loss goes to 0, that's probably not a good sign as well!

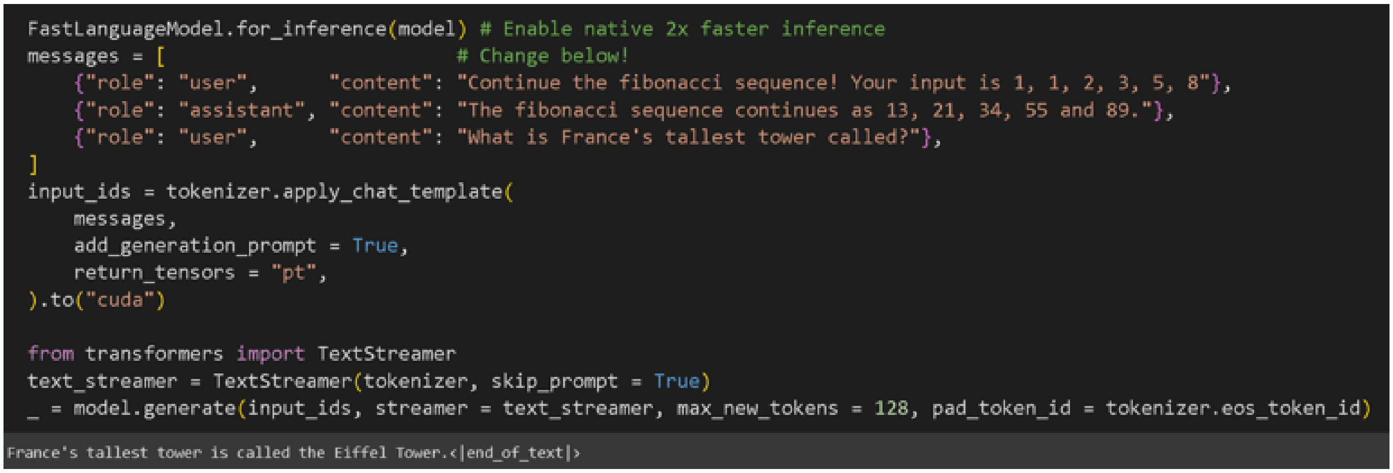
11. Inference / running the model



```python
FastLanguageModel.for_inference(model) # Enable native 2x faster inference
messages = [                    # Change below!
    {"role": "user", "content": "Continue the fibonacci sequence! Your input is 1, 1, 2, 3, 5, 8,"},
]
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)
```

```
The next number in the Fibonacci sequence is 13.<|end_of_text|>
```

Now let's run the model after we completed the training process! You can edit the yellow underlined part! In fact, because we created a multi turn chatbot, we can now also call the model as if it saw some conversations in the past like below:



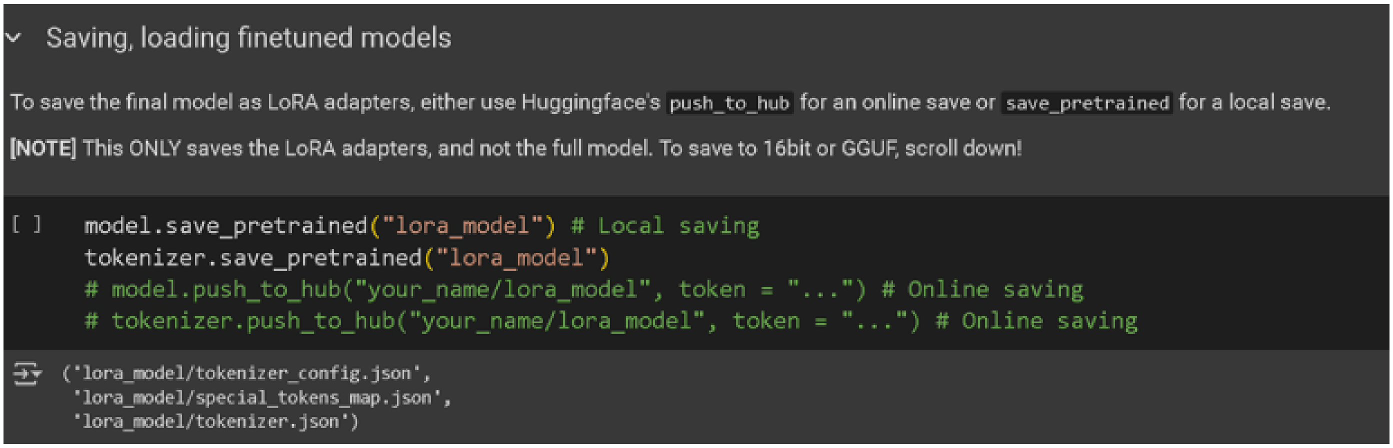
```python
FastLanguageModel.for_inference(model) # Enable native 2x faster inference
messages = [                                    # Change below!
    {"role": "user",      "content": "Continue the fibonacci sequence! Your input is 1, 1, 2, 3, 5, 8"},
    {"role": "assistant", "content": "The fibonacci sequence continues as 13, 21, 34, 55 and 89."},
    {"role": "user",      "content": "What is France's tallest tower called?"},
]
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)
```

```
France's tallest tower is called the Eiffel Tower.<|end_of_text|>
```

Reminder Unsloth itself provides **2x faster inference** natively as well, so always do not forget to call `FastLanguageModel.for_inference(model)`. If you want the model to output longer responses, set `max_new_tokens = 128` to some larger number like 256 or 1024. Notice you will have to wait longer for the result as well!

12. Saving the model

We can now save the finetuned model as a small 100MB file called a LoRA adapter like below. You can instead push to the Hugging Face hub as well if you want to upload your model! Remember to get a Hugging Face token via https://huggingface.co/settings/tokens and add your token!



Saving, loading finetuned models

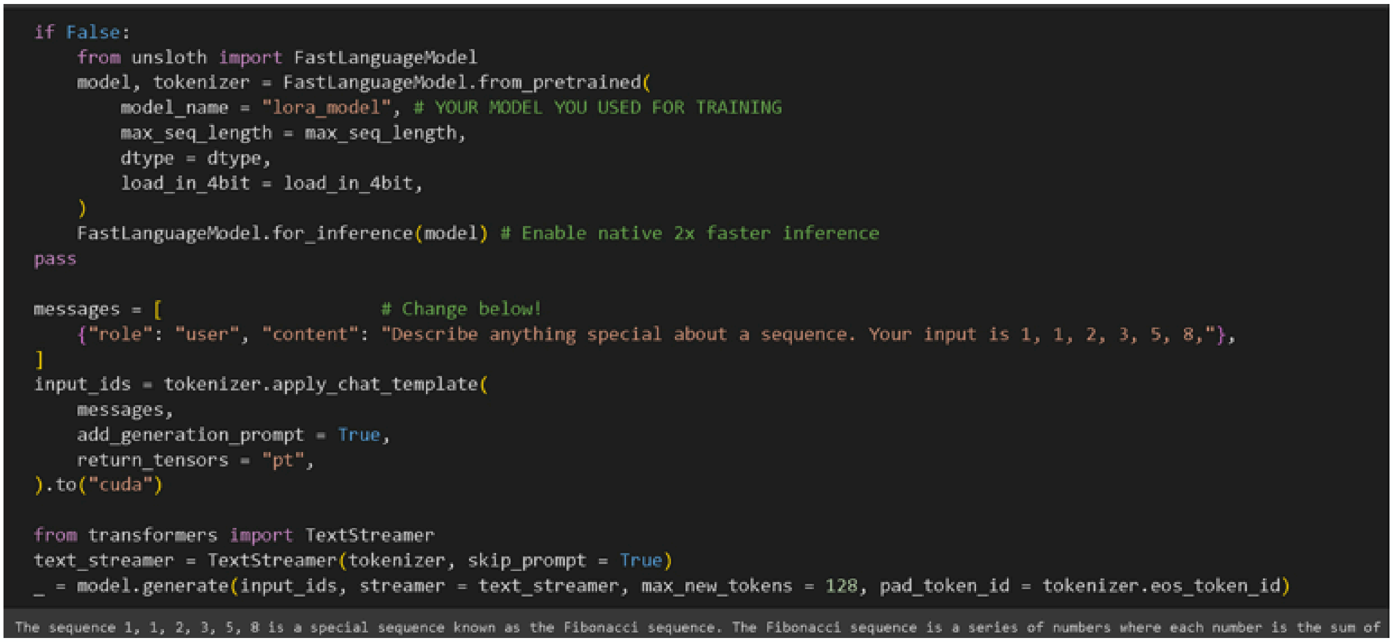
To save the final model as LoRA adapters, either use Huggingface's `push_to_hub` for an online save or `save_pretrained` for a local save.

[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')
```

After saving the model, we can again use Unsloth to run the model itself! Use `FastLanguageModel` again to call it for inference!

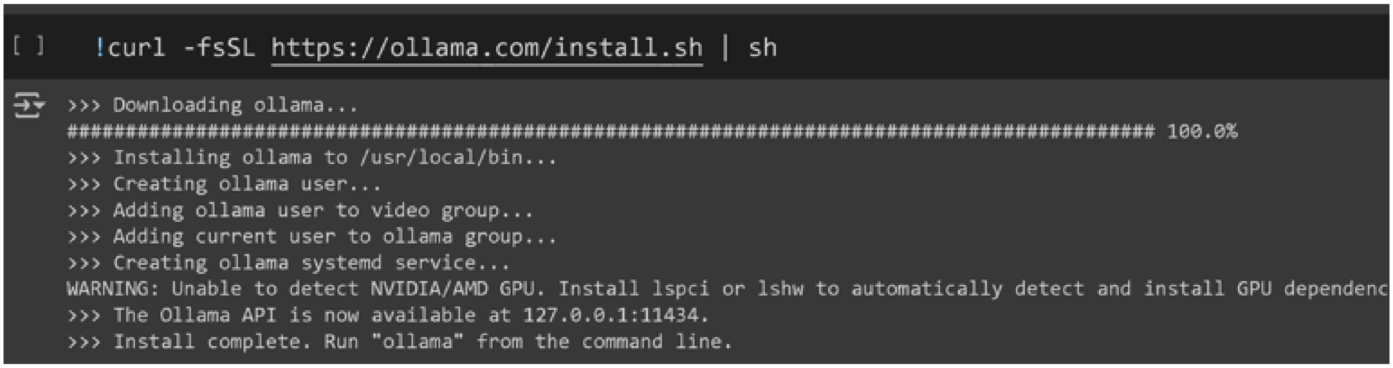


```
if False:
    from unsloth import FastLanguageModel
    model, tokenizer = FastLanguageModel.from_pretrained(
        model_name = "lora_model", # YOUR MODEL YOU USED FOR TRAINING
        max_seq_length = max_seq_length,
        dtype = dtype,
        load_in_4bit = load_in_4bit,
    )
    FastLanguageModel.for_inference(model)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

messages = [                    # Change below!
    {"role": "user", "content": "Describe anything special about a sequence. Your input is 1, 1, 2, 3, 5, 8,"},
]
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)
```

```
The sequence 1, 1, 2, 3, 5, 8 is a special sequence known as the Fibonacci sequence. The Fibonacci sequence is a series of numbers where each number is the sum of
```

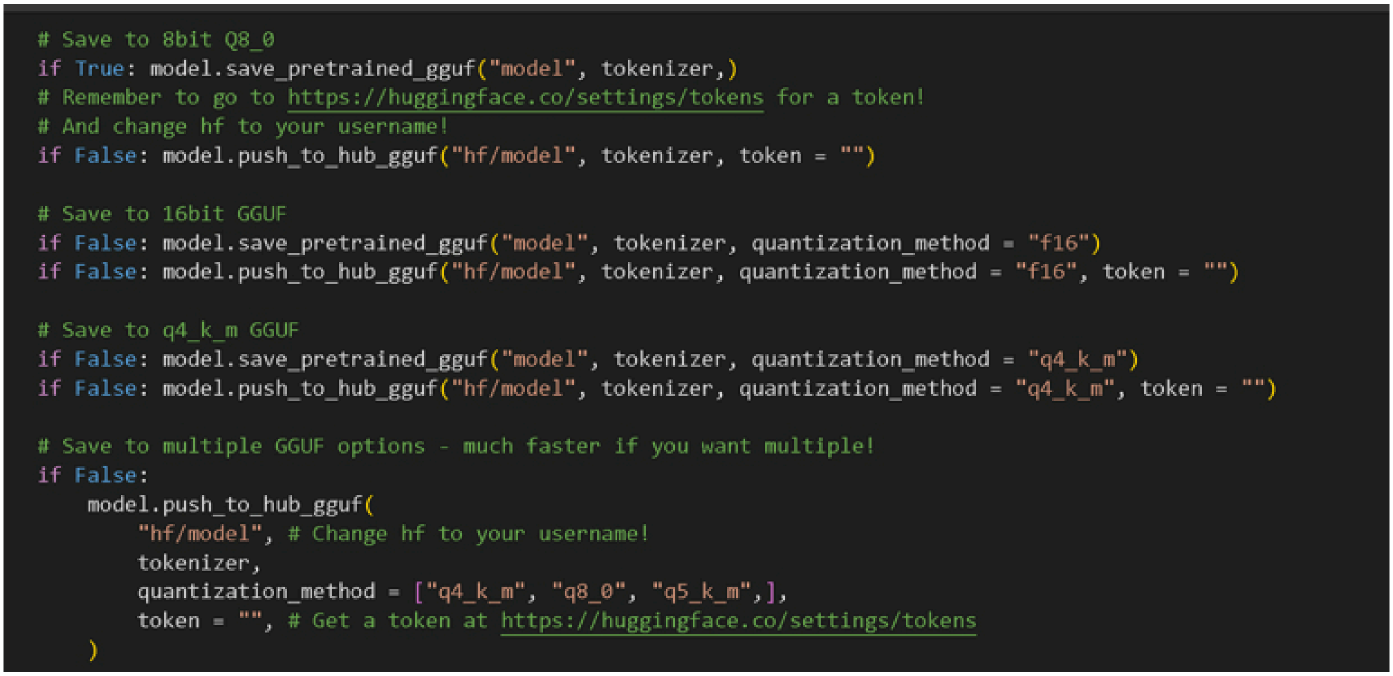
13. Exporting to Ollama

Finally we can export our finetuned model to Ollama itself! First we have to install Ollama in the Colab notebook:



```
[ ]  !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 dependenc
>>> The Ollama API is now available at 127.0.0.1:11434.
>>> Install complete. Run "ollama" from the command line.
```

Then we export the finetuned model we have to llama.cpp's GGUF formats like below:

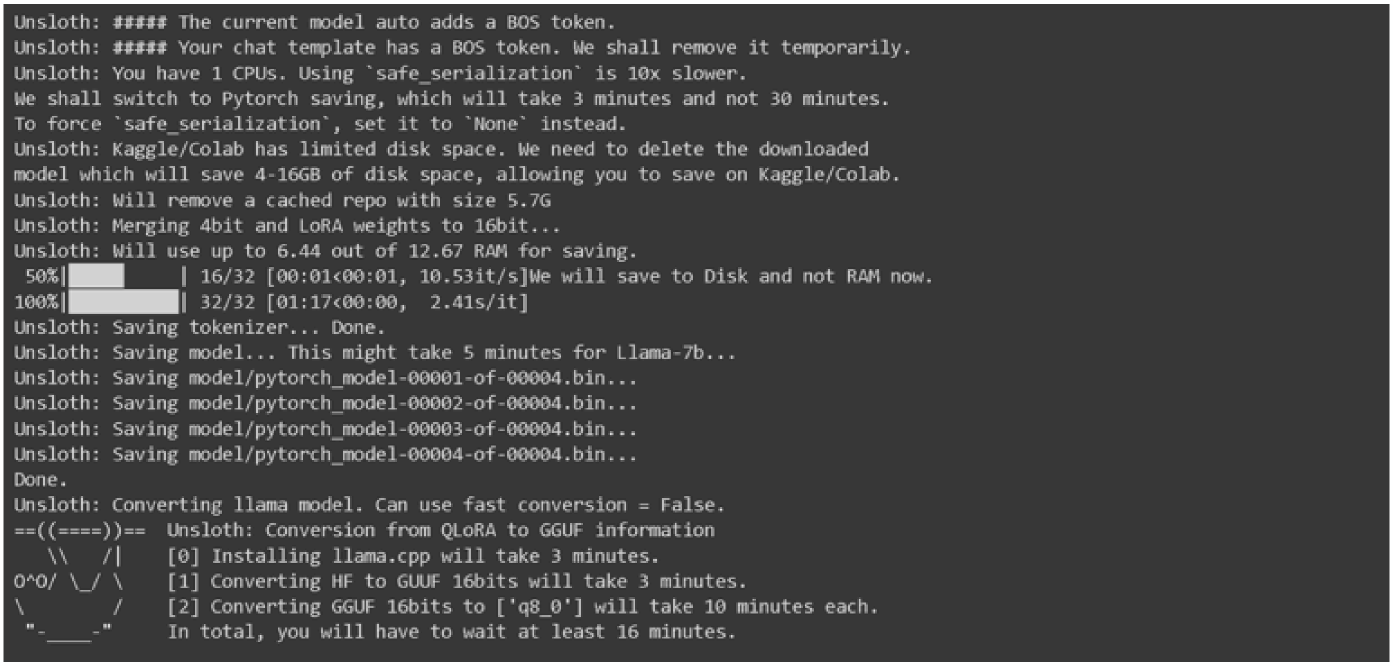


```python
# Save to 8bit Q8_0
if True: model.save_pretrained_gguf("model", tokenizer,)
# Remember to go to https://huggingface.co/settings/tokens for a token!
# And change hf to your username!
if False: model.push_to_hub_gguf("hf/model", tokenizer, token = "")

# 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
    )
```

Reminder to convert `False` to `True` for 1 row, and not change every row to `True`, or else you'll be waiting for a very time! We normally suggest the first row getting set to `True`, so we can export the finetuned model quickly to `Q8_0` format (8 bit quantization). We also allow you to export to a whole list of quantization methods as well, with a popular one being `q4_k_m`.

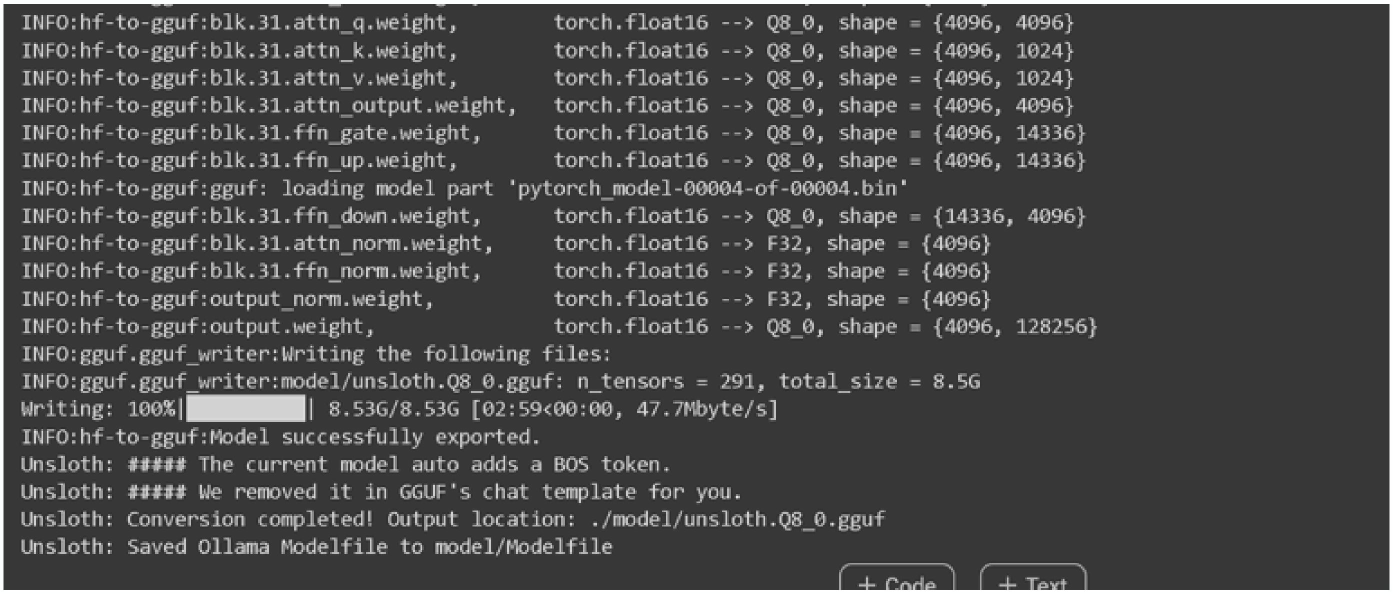
Head over to https://github.com/ggerganov/llama.cpp to learn more about GGUF. We also have some manual instructions of how to export to GGUF if you want here: https://github.com/unslothai/unsloth/wiki#manually-saving-to-gguf

You will see a long list of text like below - please wait 5 to 10 minutes!!



```
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 6.44 out of 12.67 RAM for saving.
 50%|█████     | 16/32 [00:01<00:01, 10.53it/s]We will save to Disk and not RAM now.
100%|██████████| 32/32 [01:17<00:00,  2.41s/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.
```

And finally at the very end, it'll look like below:



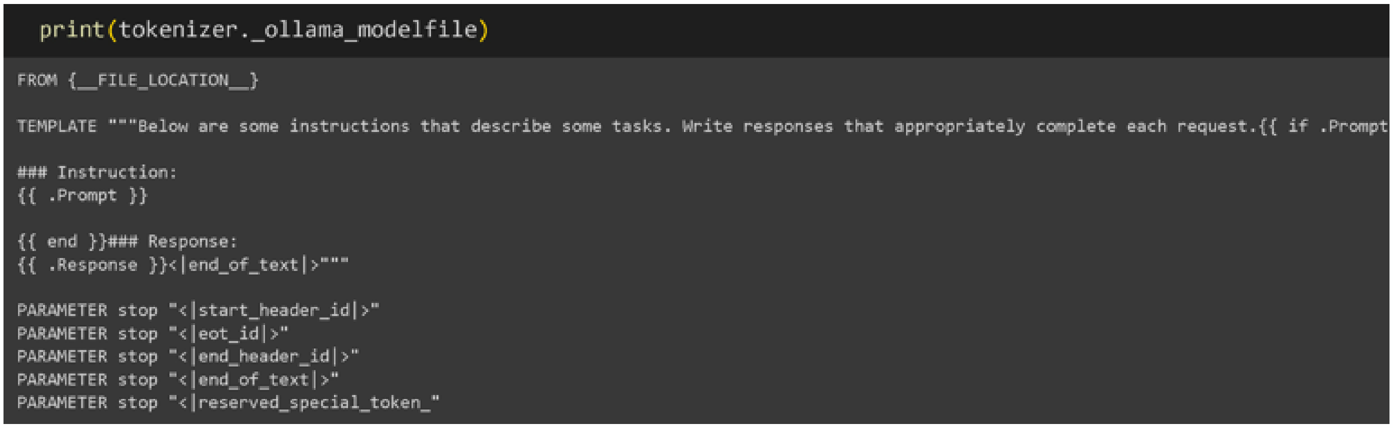
```
INFO:hf-to-gguf:blk.31.attn_q.weight,        torch.float16 --> Q8_0, shape = {4096, 4096}
INFO:hf-to-gguf:blk.31.attn_k.weight,        torch.float16 --> Q8_0, shape = {4096, 1024}
INFO:hf-to-gguf:blk.31.attn_v.weight,        torch.float16 --> Q8_0, shape = {4096, 1024}
INFO:hf-to-gguf:blk.31.attn_output.weight,   torch.float16 --> Q8_0, shape = {4096, 4096}
INFO:hf-to-gguf:blk.31.ffn_gate.weight,      torch.float16 --> Q8_0, shape = {4096, 14336}
INFO:hf-to-gguf:blk.31.ffn_up.weight,        torch.float16 --> Q8_0, shape = {4096, 14336}
INFO:hf-to-gguf:gguf: loading model part 'pytorch_model-00004-of-00004.bin'
INFO:hf-to-gguf:blk.31.ffn_down.weight,      torch.float16 --> Q8_0, shape = {14336, 4096}
INFO:hf-to-gguf:blk.31.attn_norm.weight,     torch.float16 --> F32, shape = {4096}
INFO:hf-to-gguf:blk.31.ffn_norm.weight,      torch.float16 --> F32, shape = {4096}
INFO:hf-to-gguf:output_norm.weight,          torch.float16 --> F32, shape = {4096}
INFO:hf-to-gguf:output.weight,               torch.float16 --> Q8_0, shape = {4096, 128256}
INFO:gguf.gguf_writer:Writing the following files:
INFO:gguf.gguf_writer:model/unsloth.Q8_0.gguf: n_tensors = 291, total_size = 8.5G
Writing: 100%|██████████| 8.53G/8.53G [02:59<00:00, 47.7Mbyte/s]
INFO:hf-to-gguf:Model successfully exported.
Unsloth: ##### The current model auto adds a BOS token.
Unsloth: ##### We removed it in GGUF's chat template for you.
Unsloth: Conversion completed! Output location: ./model/unsloth.Q8_0.gguf
Unsloth: Saved Ollama Modelfile to model/Modelfile
```

Then, we have to run Ollama itself in the background. We use `subprocess` because Colab doesn't like asynchronous calls, but normally one just runs `ollama serve` in the terminal / command prompt.

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

14. Automatic `Modelfile` creation

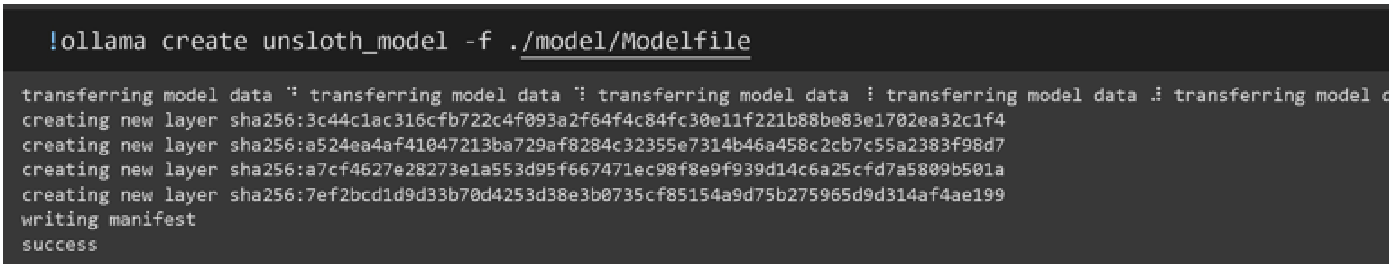
The trick Unsloth provides is we automatically create a `Modelfile` which Ollama requires! This is a just a list of settings and includes the chat template which we used for the finetune process! You can also print the `Modelfile` generated like below:



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

```
FROM {__FILE_LOCATION__}

TEMPLATE """Below are some instructions that describe some tasks. Write responses that appropriately complete each request.{{ if .Prompt

### Instruction:
{{ .Prompt }}

{{ end }}### Response:
{{ .Response }}<|end_of_text|>"""

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

We then ask Ollama to create a model which is Ollama compatible, by using the `Modelfile`

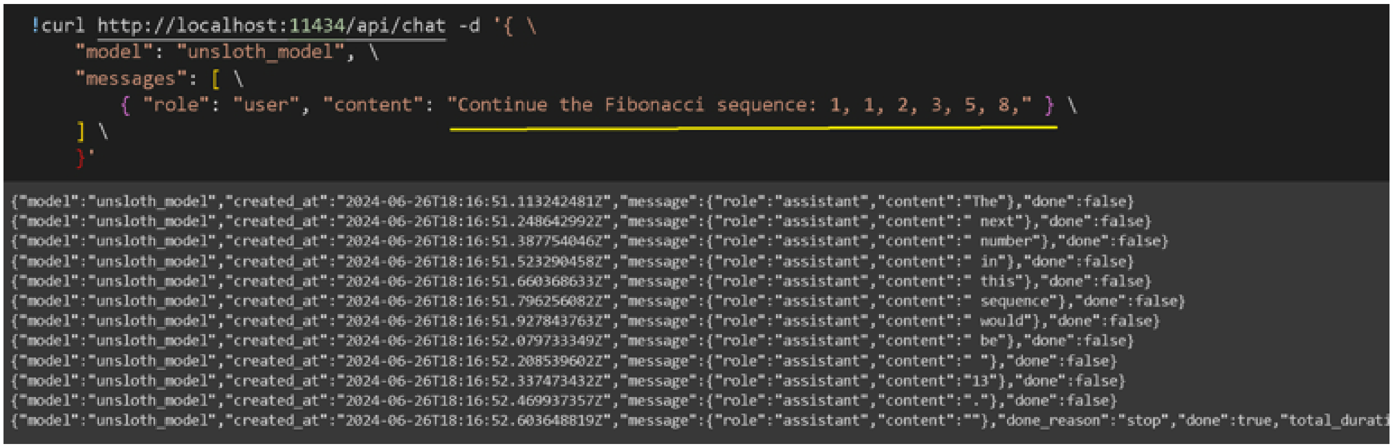


```
!ollama create unsloth_model -f ./model/Modelfile
```

```
transferring model data ⠙ transferring model data ⠹ transferring model data ⠸ transferring model data ⠼ transferring model d
creating new layer sha256:3c44c1ac316cfb722c4f093a2f64f4c84fc30e11f221b88be83e1702ea32c1f4
creating new layer sha256:a524ea4af41047213ba729af8284c32355e7314b46a458c2cb7c55a2383f98d7
creating new layer sha256:a7cf4627e28273e1a553d95f667471ec98f8e9f939d14c6a25cfd7a5809b501a
creating new layer sha256:7ef2bcd1d9d33b70d4253d38e3b0735cf85154a9d75b275965d9d314af4ae199
writing manifest
success
```

15. Ollama Inference

And we can now call the model for inference if you want to do call the Ollama server itself which is running on your own local machine / in the free Colab notebook in the background. Remember you can edit the yellow underlined part.

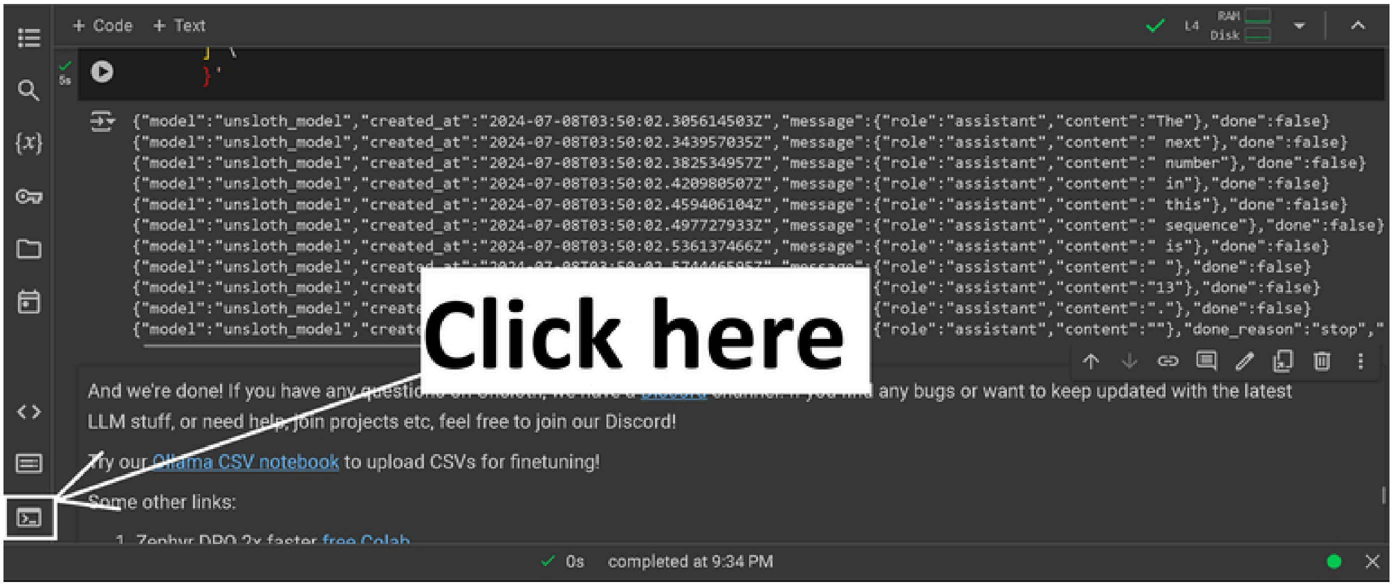


```
!curl http://localhost:11434/api/chat -d '{ \
    "model": "unsloth_model", \
    "messages": [ \
        { "role": "user", "content": "Continue the Fibonacci sequence: 1, 1, 2, 3, 5, 8," } \
    ] \
    }'
```

```
{"model":"unsloth_model","created_at":"2024-06-26T18:16:51.113242481Z","message":{"role":"assistant","content":"The"},"done":false}
{"model":"unsloth_model","created_at":"2024-06-26T18:16:51.248642992Z","message":{"role":"assistant","content":" next"},"done":false}
{"model":"unsloth_model","created_at":"2024-06-26T18:16:51.387754046Z","message":{"role":"assistant","content":" number"},"done":false}
{"model":"unsloth_model","created_at":"2024-06-26T18:16:51.523290458Z","message":{"role":"assistant","content":" in"},"done":false}
{"model":"unsloth_model","created_at":"2024-06-26T18:16:51.660368633Z","message":{"role":"assistant","content":" this"},"done":false}
{"model":"unsloth_model","created_at":"2024-06-26T18:16:51.796256082Z","message":{"role":"assistant","content":" sequence"},"done":false}
{"model":"unsloth_model","created_at":"2024-06-26T18:16:51.927843763Z","message":{"role":"assistant","content":" would"},"done":false}
{"model":"unsloth_model","created_at":"2024-06-26T18:16:52.079733349Z","message":{"role":"assistant","content":" be"},"done":false}
{"model":"unsloth_model","created_at":"2024-06-26T18:16:52.208539602Z","message":{"role":"assistant","content":" "},"done":false}
{"model":"unsloth_model","created_at":"2024-06-26T18:16:52.337473432Z","message":{"role":"assistant","content":"13"},"done":false}
{"model":"unsloth_model","created_at":"2024-06-26T18:16:52.469937357Z","message":{"role":"assistant","content":"."},"done":false}
{"model":"unsloth_model","created_at":"2024-06-26T18:16:52.603648819Z","message":{"role":"assistant","content":""},"done_reason":"stop","done":true,"total_durati
```

16. Interactive ChatGPT style

But to actually run the finetuned model like a ChatGPT, we have to do a bit more! First click the terminal icon and a Terminal will pop up. It's on the left sidebar.



```
{"model":"unsloth_model","created_at":"2024-07-08T03:50:02.305614503Z","message":{"role":"assistant","content":"The"},"done":false}
{"model":"unsloth_model","created_at":"2024-07-08T03:50:02.343957035Z","message":{"role":"assistant","content":" next"},"done":false}
{"model":"unsloth_model","created_at":"2024-07-08T03:50:02.382534957Z","message":{"role":"assistant","content":" number"},"done":false}
{"model":"unsloth_model","created_at":"2024-07-08T03:50:02.420980507Z","message":{"role":"assistant","content":" in"},"done":false}
{"model":"unsloth_model","created_at":"2024-07-08T03:50:02.459406104Z","message":{"role":"assistant","content":" this"},"done":false}
{"model":"unsloth_model","created_at":"2024-07-08T03:50:02.497727933Z","message":{"role":"assistant","content":" sequence"},"done":false}
{"model":"unsloth_model","created_at":"2024-07-08T03:50:02.536137466Z","message":{"role":"assistant","content":" is"},"done":false}
{"model":"unsloth_model","create                                      {"role":"assistant","content":" "},"done":false}
{"model":"unsloth_model","create                                      {"role":"assistant","content":"13"},"done":false}
{"model":"unsloth_model","create                                      {"role":"assistant","content":"."},"done":false}
{"model":"unsloth_model","create                                      {"role":"assistant","content":""},"done_reason":"stop","
```

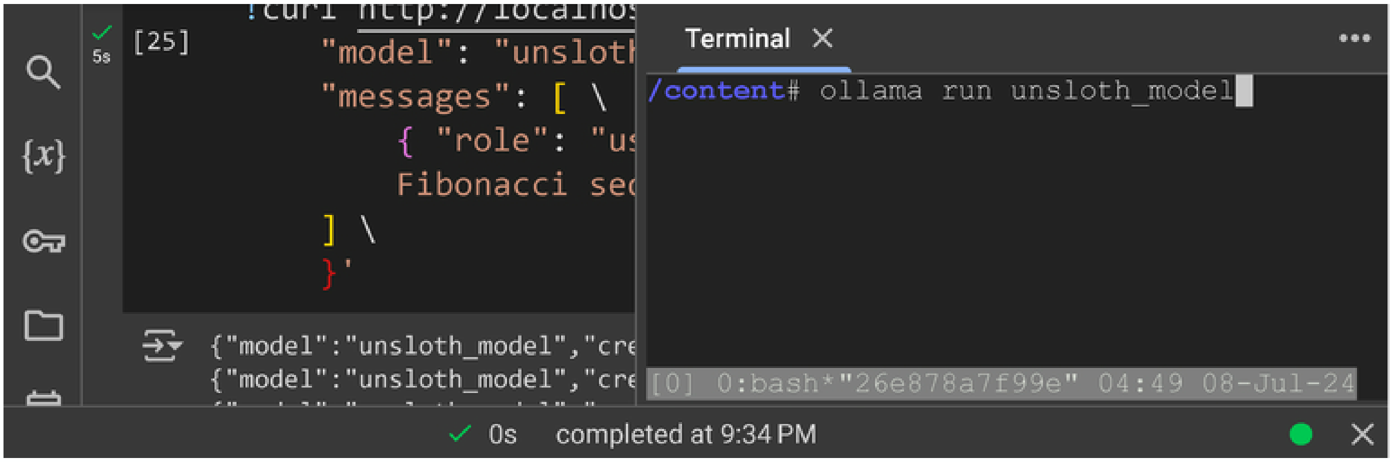
And we're done! If you have any questio   any bugs or want to keep updated with the latest LLM stuff, or need help, join projects etc, feel free to join our Discord!

Try our Ollama CSV notebook to upload CSVs for finetuning!

Some other links:

✓ 0s completed at 9:34 PM

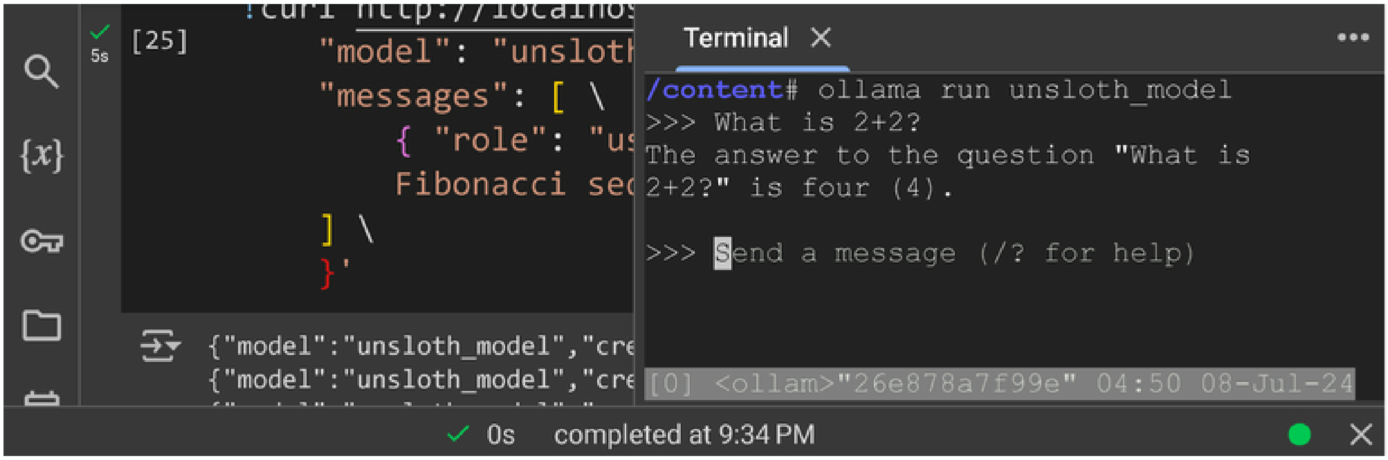
Then, you might have to press ENTER twice to remove some weird output in the Terminal window. Wait a few seconds and type `ollama run unsloth_model` then hit ENTER.



```
[25]        "model": "unslotl
            "messages": [ \
                { "role": "us
                Fibonacci sec
            ] \
            }'
    {"model":"unsloth_model","cre
    {"model":"unsloth_model","cre
```

```
/content# ollama run unsloth_model
[0] 0:bash*"26e878a7f99e" 04:49 08-Jul-24
```

And finally, you can interact with the finetuned model just like an actual ChatGPT! Hit CTRL + D to exit the system, and hit ENTER to converse with the chatbot!



```
[25]        "model": "unslotl
            "messages": [ \
                { "role": "us
                Fibonacci sec
            ] \
            }'
    {"model":"unsloth_model","cre
    {"model":"unsloth_model","cre
```

```
/content# ollama run unsloth_model
>>> What is 2+2?
The answer to the question "What is
2+2?" is four (4).

>>> Send a message (/? for help)
[0] <ollam>"26e878a7f99e" 04:50 08-Jul-24
```

You've done it!

You've successfully finetuned a language model and exported it to Ollama with Unsloth 2x faster and with 70% less VRAM! And all this for free in a Google Colab notebook!

If you want to learn how to do reward modelling, do continued pretraining, export to vLLM or GGUF, do text completion, or learn more about finetuning tips and tricks, head over to our Github.

If you need any help on finetuning, you can also join our Discord server here. If you want help with Ollama, you can also join their server here.

And finally, we want to thank you for reading and following this far! We hope this made you understand some of the nuts and bolts behind finetuning language models, and we hope this was useful!

To access our Alpaca dataset example click here, and our CSV / Excel finetuning guide is here.

Previous
DeepSeek-R1 Dynamic 1.58-bit

Next
Datasets 101

Last updated 4 days ago

Was this helpful?