```
%%capture
!pip install unsloth "xformers==0.0.28.post2"
# Also get the latest nightly Unsloth!
!pip uninstall unsloth -y && pip install --upgrade --no-cache-dir "unsloth[colab-new] @ git+https://github.com/unslothai/unsloth
```

```
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/Meta-Llama-3.1-8B-bnb-4bit",      # Llama-3.1 2x faster
    "unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit",
    "unsloth/Meta-Llama-3.1-70B-bnb-4bit",
    "unsloth/Meta-Llama-3.1-405B-bnb-4bit",    # 4bit for 405b!
    "unsloth/Mistral-Small-Instruct-2409",     # Mistral 22b 2x faster!
    "unsloth/mistral-7b-instruct-v0.3-bnb-4bit",
    "unsloth/Phi-3.5-mini-instruct",           # Phi-3.5 2x faster!
    "unsloth/Phi-3-medium-4k-instruct",
    "unsloth/gemma-2-9b-bnb-4bit",
    "unsloth/gemma-2-27b-bnb-4bit",            # Gemma 2x faster!

    "unsloth/Llama-3.2-1B-bnb-4bit",           # NEW! Llama 3.2 models
    "unsloth/Llama-3.2-1B-Instruct-bnb-4bit",
    "unsloth/Llama-3.2-3B-bnb-4bit",
    "unsloth/Llama-3.2-3B-Instruct-bnb-4bit",
] # More models at https://huggingface.co/unsloth

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/Llama-3.2-3B-Instruct", # or choose "unsloth/Llama-3.2-1B-Instruct"
    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.
==((====))==  Unsloth 2024.11.7: Fast Llama patching. Transformers = 4.46.2.
   \\   /|    GPU: Tesla T4. Max memory: 14.748 GB. Platform = Linux.
O^O/ \_/ \    Pytorch: 2.5.0+cu121. CUDA = 7.5. CUDA Toolkit = 12.1.
\        /    Bfloat16 = FALSE. FA [Xformers = 0.0.28.post2. FA2 = False]
 "-____-"     Free Apache license: http://github.com/unslothai/unsloth
Unsloth: Fast downloading is enabled - ignore downloading bars which are red colored!
model.safetensors: 100%          2.24G/2.24G [00:14<00:00, 379MB/s]
generation_config.json: 100%     184/184 [00:00<00:00, 7.05kB/s]
tokenizer_config.json: 100%      54.6k/54.6k [00:00<00:00, 674kB/s]
tokenizer.json: 100%             9.09M/9.09M [00:00<00:00, 41.0MB/s]
special_tokens_map.json: 100%    454/454 [00:00<00:00, 34.3kB/s]
```

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

```
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/Llama-3.2-1B-Instruct", # or choose "unsloth/Llama-3.2-3B-Instruct"
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
    # token = "hf_...", # comenting it
)
```

```
==((====))==  Unsloth 2024.11.7: Fast Llama patching. Transformers = 4.46.2.
   \\   /|    GPU: Tesla T4. Max memory: 14.748 GB. Platform = Linux.
O^O/ \_/ \    Pytorch: 2.5.0+cu121. CUDA = 7.5. CUDA Toolkit = 12.1.
\        /    Bfloat16 = FALSE. FA [Xformers = 0.0.28.post2. FA2 = False]
 "-____-"     Free Apache license: http://github.com/unslothai/unsloth
Unsloth: Fast downloading is enabled - ignore downloading bars which are red colored!
model.safetensors: 100%                         1.03G/1.03G [00:17<00:00, 44.2MB/s]
generation_config.json: 100%                    184/184 [00:00<00:00, 9.51kB/s]
tokenizer_config.json: 100%                     54.6k/54.6k [00:00<00:00, 3.23MB/s]
tokenizer.json: 100%                            9.09M/9.09M [00:00<00:00, 35.1MB/s]
special_tokens_map.json: 100%                   454/454 [00:00<00:00, 35.3kB/s]
```

```
model = FastLanguageModel.get_peft_model(
    model,
    r = 16, #  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 2024.11.7 patched 16 layers with 16 QKV layers, 16 O layers and 16 MLP layers.
```

Data Prep

We now use the `Llama-3.1` format for conversation style finetunes. We use [Maxime Labonne's FineTome-100k](#) dataset in ShareGPT style. But we convert it to HuggingFace's normal multiturn format `("role", "content")` instead of `("from", "value")` / Llama-3 renders multi turn conversations like below:

```
<|begin_of_text|><|start_header_id|>user<|end_header_id|>

Hello!<|eot_id|><|start_header_id|>assistant<|end_header_id|>

Hey there! How are you?<|eot_id|><|start_header_id|>user<|end_header_id|>

I'm great thanks!<|eot_id|>
```

We use our `get_chat_template` function to get the correct chat template. We support `zephyr, chatml, mistral, llama, alpaca, vicuna, vicuna_old, phi3, llama3` and more.

```
from unsloth.chat_templates import get_chat_template

tokenizer = get_chat_template(
    tokenizer,
    chat_template = "llama-3.1",
)

def formatting_prompts_func(examples):
    convos = examples["conversations"]
    texts = [tokenizer.apply_chat_template(convo, tokenize = False, add_generation_prompt = False) for convo in convos]
    return { "text" : texts, }
pass



# load a my private dataset(.csv file) in hf in Siddartha-49/college-advisor repo
```

```
!huggingface-cli login
```

```
    _|    _|  _|    _|    _|_|_|    _|_|_|  _|_|_|  _|      _|    _|_|_|      _|_|_|_|    _|_|      _|_|_|  _|_|_|_|
    _|    _|  _|    _|  _|        _|          _|    _|_|    _|  _|            _|        _|    _|  _|        _|
    _|_|_|_|  _|    _|  _|  _|_|  _|  _|_|    _|    _|  _|  _|  _|  _|_|      _|_|_|    _|_|_|_|  _|        _|_|_|
    _|    _|  _|    _|  _|    _|  _|    _|    _|    _|    _|_|  _|    _|      _|        _|    _|  _|        _|
    _|    _|    _|_|      _|_|_|    _|_|_|  _|_|_|  _|      _|    _|_|_|      _|        _|    _|    _|_|_|  _|_|_|_|

    To log in, `huggingface_hub` requires a token generated from https://huggingface.co/settings/tokens .
Enter your token (input will not be visible):
Add token as git credential? (Y/n) Y
Token is valid (permission: fineGrained).
The token `ragathon` has been saved to /root/.cache/huggingface/stored_tokens
Cannot authenticate through git-credential as no helper is defined on your machine.
You might have to re-authenticate when pushing to the Hugging Face Hub.
Run the following command in your terminal in case you want to set the 'store' credential helper as default.

git config --global credential.helper store

Read https://git-scm.com/book/en/v2/Git-Tools-Credential-Storage for more details.
Token has not been saved to git credential helper.
Your token has been saved to /root/.cache/huggingface/token
Login successful.
The current active token is: `ragathon`
```

```
from datasets import load_dataset
dataset1 = load_dataset("Siddartha-49/college-advisor", split = "train")
```

```
conversations_dataset.parquet: 100%                                21.6k/21.6k [00:00<00:00, 489kB/s]

Generating train split:    258/0 [00:00<00:00, 4973.16 examples/s]
```

```
type(dataset1)
```

```
datasets.arrow_dataset.Dataset
def __init__(arrow_table: Table, info: Optional[DatasetInfo]=None, split:
Optional[NamedSplit]=None, indices_table: Optional[Table]=None, fingerprint:
Optional[str]=None)

A Dataset backed by an Arrow table.
```

```
dataset1
```

```
Dataset({
    features: ['conversations', 'source'],
    num_rows: 258
})
```

```
dataset1[1]
```

```
{'conversations': [{'from': 'human',
   'value': 'What are the required specialization core classes for the MS in Software Engineering - Enterprise Software
Technologies program?'},
  {'from': 'gpt',
   'value': 'The required specialization core classes are CMPE 273 Enterprise Distributed Systems and CMPE 275 Enterprise
Application Development, totaling 6 units.'}],
 'source': 'Siddartha-49'}
```

```
dataset1[1]["conversations"]
```

```
[{'from': 'human',
  'value': 'What are the required specialization core classes for the MS in Software Engineering - Enterprise Software
Technologies program?'},
 {'from': 'gpt',
  'value': 'The required specialization core classes are CMPE 273 Enterprise Distributed Systems and CMPE 275 Enterprise
Application Development, totaling 6 units.'}]
```

```
type(dataset1[1]["conversations"])
```

```
list
```

```
type(dataset1[1]["conversations"][0])
```

```
dict
```

```
dataset_root = load_dataset(
    "Siddartha-49/college-advisor",
    data_files="advisor-200.csv",
    split = "train"
)
```

```
dataset_root
```

```
Dataset({
    features: ['[{"from": "human", "value": "What is the MS in Software Engineering - Enterprise Software Technologies
program about?"}, {"from": "gpt", "value": "The MS in Software Engineering - Enterprise Software Technologies program
prepares students to be technical leaders in software development, focusing on distributed N-Tier Client/Server
architectures and the latest technologies and trends in Enterprise software development."}]'],
    num_rows: 257
})
```

```
from datasets import load_dataset
dataset = load_dataset("mlabonne/FineTome-100k", split = "train")
```

```
type(dataset)
```

```
datasets.arrow_dataset.Dataset
def __init__(arrow_table: Table, info: Optional[DatasetInfo]=None, split:
Optional[NamedSplit]=None, indices_table: Optional[Table]=None, fingerprint:
Optional[str]=None)

A Dataset backed by an Arrow table.
```

```
dataset
```

```
Dataset({
    features: ['conversations', 'source', 'score'],
    num_rows: 100000
})
```

```
dataset[1]
```

```
{'conversations': [{'from': 'human',
   'value': 'Explain how recursion works and provide a recursive function in Python that calculates the factorial of a
given number.'},
  {'from': 'gpt',
   'value': "Recursion is a programming technique where a function calls itself to solve a problem. It breaks down a
complex problem into smaller, more manageable subproblems until a base case is reached. The base case is a condition where
the function does not call itself, but instead returns a specific value or performs a specific action.\n\nIn the case of
calculating the factorial of a number, recursion can be used to break down the problem into simpler subproblems. The
factorial of a non-negative integer n is the product of all positive integers less than or equal to n.\n\nHere is a
recursive function in Python that calculates the factorial of a given number:\n\n```python\ndef factorial(n):\n    # Base
case: factorial of 0 or 1 is 1\n    if n == 0 or n == 1:\n        return 1\n    # Recursive case: factorial of n is n
multiplied by factorial of (n-1)\n    else:\n        return n * factorial(n - 1)\n```\n\nIn this function, the base case is
when n equals 0 or 1, as the factorial of these numbers is defined as 1. For any other positive integer, the function calls
itself with a smaller value (n-1) and multiplies the result by n.\n\nFor example, let's calculate the factorial of 5 using
this function:\n\n```python\nprint(factorial(5))\n```\n\nThe function will execute as follows:\n1. factorial(5) calls
factorial(4) and multiplies the result by 5.\n2. factorial(4) calls factorial(3) and multiplies the result by 4.\n3.
factorial(3) calls factorial(2) and multiplies the result by 3.\n4. factorial(2) calls factorial(1) and multiplies the
result by 2.\n5. factorial(1) is a base case and returns 1.\n6. The final result is calculated as: 5 * 4 * 3 * 2 * 1 =
1"}],
 'source': 'infini-instruct-top-500k',
 'score': 5.157649040222168}
```

```
dataset[1]["conversations"]
```

```
[{'from': 'human',
  'value': 'Explain how recursion works and provide a recursive function in Python that calculates the factorial of a given
number.'},
 {'from': 'gpt',
  'value': "Recursion is a programming technique where a function calls itself to solve a problem. It breaks down a complex
problem into smaller, more manageable subproblems until a base case is reached. The base case is a condition where the
function does not call itself, but instead returns a specific value or performs a specific action.\n\nIn the case of
calculating the factorial of a number, recursion can be used to break down the problem into simpler subproblems. The
factorial of a non-negative integer n is the product of all positive integers less than or equal to n.\n\nHere is a
recursive function in Python that calculates the factorial of a given number:\n\n```python\ndef factorial(n):\n    # Base
case: factorial of 0 or 1 is 1\n    if n == 0 or n == 1:\n        return 1\n    # Recursive case: factorial of n is n
multiplied by factorial of (n-1)\n    else:\n        return n * factorial(n - 1)\n```\n\nIn this function, the base case is
when n equals 0 or 1, as the factorial of these numbers is defined as 1. For any other positive integer, the function calls
```

```
itself with a smaller value (n-1) and multiplies the result by n.\n\nFor example, let's calculate the factorial of 5 using
this function:\n\n```python\nprint(factorial(5))\n```\n\nThe function will execute as follows:\n1. factorial(5) calls
factorial(4) and multiplies the result by 5.\n2. factorial(4) calls factorial(3) and multiplies the result by 4.\n3.
factorial(3) calls factorial(2) and multiplies the result by 3.\n4. factorial(2) calls factorial(1) and multiplies the
result by 2.\n5. factorial(1) is a base case and returns 1.\n6. The final result is calculated as: 5 * 4 * 3 * 2 * 1 = 1"}]
```

```
type(dataset[1]["conversations"])
```

```
list
```

```
type(dataset[1]["conversations"][0])
```

```
dict
```

```
# drop source and score columns from dataset
dataset = dataset.remove_columns(["source", "score"])
```

```
dataset[1]
```

```
{'conversations': '[{"from": "human", "value": "What are the required specialization core classes for the MS in Software
Engineering - Enterprise Software Technologies program?"}, {"from": "gpt", "value": "The required specialization core
classes are CMPE 273 Enterprise Distributed Systems and CMPE 275 Enterprise Application Development, totaling 6 units."}]'}
```

We now use `standardize_sharegpt` to convert ShareGPT style datasets into HuggingFace's generic format. This changes the dataset from looking like:

```
{"from": "system", "value": "You are an assistant"}
{"from": "human", "value": "What is 2+2?"}
{"from": "gpt", "value": "It's 4."}
```

to

```
{"role": "system", "content": "You are an assistant"}
{"role": "user", "content": "What is 2+2?"}
{"role": "assistant", "content": "It's 4."}
```

```
from datasets import load_dataset
dataset = load_dataset("Siddartha-49/college-advisor", split = "train")
```

```
from unsloth.chat_templates import standardize_sharegpt
dataset = standardize_sharegpt(dataset)
dataset = dataset.map(formatting_prompts_func, batched = True,)
```

We look at how the conversations are structured for item 5:

```
dataset[5]["conversations"]
```

```
[{'content': 'Can I take a specialization choice class that is not listed?',
  'role': 'user'},
 {'content': 'No, you must choose from the listed specialization choice classes, CMPE 281, CMPE 283, CMPE 285, CMPE 287,
CMPE 206, CMPE 207, CMPE 257, CMPE 258, CMPE 209, and CMPE 279.',
  'role': 'assistant'}]
```

And we see how the chat template transformed these conversations.

[Notice] Llama 3.1 Instruct's default chat template default adds `"Cutting Knowledge Date: December 2023\nToday Date: 26 July 2024"`, so do not be alarmed!

```
dataset[5]["text"]
```

```
'<|begin_of_text|><|start_header_id|>system<|end_header_id|>\n\nCutting Knowledge Date: December 2023\nToday Date: 26 July
2024\n\n<|eot_id|><|start_header_id|>user<|end_header_id|>\n\nCan I take a specialization choice class that is not listed?<
|eot_id|><|start_header_id|>assistant<|end_header_id|>\n\nNo, you must choose from the listed specialization choice classe
s, CMPE 281, CMPE 283, CMPE 285, CMPE 287, CMPE 206, CMPE 207, CMPE 257, CMPE 258, CMPE 209, and CMPE 279.<|eot_id|>'
```

```
dataset[5]
```

```
{'conversations': [{'content': 'Can I take a specialization choice class that is not listed?',
   'role': 'user'},
  {'content': 'No, you must choose from the listed specialization choice classes, CMPE 281, CMPE 283, CMPE 285, CMPE 287,
CMPE 206, CMPE 207, CMPE 257, CMPE 258, CMPE 209, and CMPE 279.',
   'role': 'assistant'}],
 'source': 'Siddartha-49',
 'text': '<|begin_of_text|><|start_header_id|>system<|end_header_id|>\n\nCutting Knowledge Date: December 2023\nToday Date:
26 July 2024\n\n<|eot_id|><|start_header_id|>user<|end_header_id|>\n\nCan I take a specialization choice class that is not
listed?<|eot_id|><|start_header_id|>assistant<|end_header_id|>\n\nNo, you must choose from the listed specialization choice
classes, CMPE 281, CMPE 283, CMPE 285, CMPE 287, CMPE 206, CMPE 207, CMPE 257, CMPE 258, CMPE 209, and CMPE 279.
<|eot_id|>'}
```

Train the model

Now let's use Huggingface TRL's `SFTTrainer`! More docs here: [TRL SFT docs](https://huggingface.co/docs/trl/sft_trainer). 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, DataCollatorForSeq2Seq
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,
    data_collator = DataCollatorForSeq2Seq(tokenizer = tokenizer),
    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,
        num_train_epochs = 10, # Set this for 1 full training run.
        # 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",
        report_to = "none", # Use this for WandB etc
    ),
)
```

```
Map (num_proc=2): 100%                                   258/258 [00:03<00:00, 79.48 examples/s]
```

We also use Unsloth's `train_on_completions` method to only train on the assistant outputs and ignore the loss on the user's inputs.

```
from unsloth.chat_templates import train_on_responses_only
trainer = train_on_responses_only(
    trainer,
    instruction_part = "<|start_header_id|>user<|end_header_id|>\n\n",
    response_part = "<|start_header_id|>assistant<|end_header_id|>\n\n",
)
```

```
Map: 100%                                   258/258 [00:00<00:00, 388.14 examples/s]
```

We verify masking is actually done:

```
tokenizer.decode(trainer.train_dataset[5]["input_ids"])
```

```
'<|begin_of_text|><|start_header_id|>system<|end_header_id|>\n\nCutting Knowledge Date: December 2023\nToday Date: 26 July
2024\n\n<|eot_id|><|start_header_id|>user<|end_header_id|>\n\nCan I take a specialization choice class that is not listed?<
|eot_id|><|start_header_id|>assistant<|end_header_id|>\n\nNo, you must choose from the listed specialization choice classe
s, CMPE 281, CMPE 283, CMPE 285, CMPE 287, CMPE 206, CMPE 207, CMPE 257, CMPE 258, CMPE 209, and CMPE 279.<|eot_id|>'
```

```
space = tokenizer(" ", add_special_tokens = False).input_ids[0]
tokenizer.decode([space if x == -100 else x for x in trainer.train_dataset[5]["labels"]])
```

```
'                                             \n\nNo, you must choose from the listed specialization choice classes, CMPE
281, CMPE 283, CMPE 285, CMPE 287, CMPE 206, CMPE 207, CMPE 257, CMPE 258, CMPE 209, and CMPE 279.<|eot_id|>'
```

We can see the System and Instruction prompts are successfully masked!

Show current memory stats

```
#@title Show current memory stats
gpu_stats = torch.cuda.get_device_properties(0)
start_gpu_memory = round(torch.cuda.max_memory_reserved() / 1024 / 1024 / 1024, 3)
max_memory = round(gpu_stats.total_memory / 1024 / 1024 / 1024, 3)
print(f"GPU = {gpu_stats.name}. Max memory = {max_memory} GB.")
print(f"{start_gpu_memory} GB of memory reserved.")
```

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

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

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs = 1
   \\   /|    Num examples = 258 | Num Epochs = 10
O^O/ \_/ \    Batch size per device = 2 | Gradient Accumulation steps = 4
\        /    Total batch size = 8 | Total steps = 320
 "-____-"     Number of trainable parameters = 11,272,192
```

[320/320 05:26, Epoch 9/10]

Step	Training Loss
1	2.410400
2	2.684800
3	1.984400
4	2.023000
5	1.871100
6	1.998900
7	1.986300
8	1.826100
9	1.332500
10	1.754600
11	1.733300
12	1.657200
13	1.462300
14	1.539300
15	1.800900
16	1.261000
17	1.181400
18	1.304800
19	1.341500
20	1.596300
21	1.440000
22	1.209000
23	1.050500
24	1.121000
25	1.107800
26	0.959900
27	0.734800
28	1.616600
29	1.359600
30	1.275700
31	1.065600
32	1.326500
33	2.637800
34	1.137300
35	0.818800
36	1.486400
37	0.677600
38	1.106700
39	0.985700
40	1.250400
41	1.138700
42	0.687400

43	1.017000
44	1.033900
45	0.299800
46	0.802900
47	0.712600
48	0.781000
49	0.625500
50	0.809900
51	0.548100
52	0.850200
53	0.535400
54	1.006200
55	0.866900
56	0.749200
57	0.658100
58	0.892900
59	0.906400
60	0.892800
61	0.740000
62	0.910300
63	1.032700
64	1.126400
65	1.765800
66	0.535800
67	0.547900
68	0.674900
69	0.378800
70	0.674500
71	0.448100
72	0.275300
73	0.303500
74	0.476700
75	0.376600
76	0.219700
77	0.467900
78	0.547400
79	0.293600
80	0.294100
81	0.695100
82	0.315700
83	0.306600
84	0.477300
85	0.396700
86	0.524700
87	0.680600
88	0.446600
89	0.343200

90	0.560900
91	0.311300
92	0.610800
93	0.271600
94	0.643000
95	0.488600
96	0.410300
97	0.356600
98	0.168300
99	0.102900
100	0.219900
101	0.166000
102	0.158400
103	0.208800
104	0.194900
105	0.129200
106	0.180600
107	0.107000
108	0.199300
109	0.347500
110	0.223500
111	0.321500
112	0.351700
113	0.126100
114	0.126900
115	0.269500
116	0.406400
117	0.172500
118	0.410100
119	0.384200
120	0.171500
121	0.314300
122	0.122200
123	0.072600
124	0.473900
125	0.072600
126	0.178800
127	0.127600
128	0.106800
129	0.158600
130	0.105400
131	0.060900
132	0.169300
133	0.119000
134	0.071000
135	0.072800
136	0.072300

```
137    0.051200
138    0.158800
139    0.060500
140    0.154600
141    0.034900
142    0.103400
143    0.099800
144    0.134000
145    0.061400
146    0.145800
147    0.180900
148    0.090600
149    0.113800
150    0.130700
151    0.060600
152    0.175600
153    0.068100
154    0.081100
155    0.126400
156    0.081600
157    0.155200
158    0.175600
159    0.198300
160    0.072000
161    0.100300
162    0.057000
163    0.104200
164    0.064200
165    0.073000
166    0.068300
167    0.076900
168    0.033100
169    0.071000
170    0.033700
171    0.056400
172    0.049600
173    0.024500
174    0.102100
175    0.102500
176    0.059800
177    0.050400
178    0.062200
179    0.027800
180    0.085900
181    0.084200
182    0.060000
183    0.056000
```

184	0.101400
185	0.057600
186	0.048300
187	0.053000
188	0.091500
189	0.075000
190	0.148600
191	0.053900
192	0.065500
193	0.094100
194	0.081800
195	0.033100
196	0.014300
197	0.038300
198	0.010400
199	0.045200
200	0.013300
201	0.012400
202	0.024800
203	0.021400
204	0.032200
205	0.070500
206	0.034000
207	0.023100
208	0.032700
209	0.053400
210	0.044100
211	0.025300
212	0.025400
213	0.076800
214	0.021500
215	0.034200
216	0.034700
217	0.034700
218	0.008000
219	0.060000
220	0.042800
221	0.043200
222	0.050800
223	0.029200
224	0.025200
225	0.038700
226	0.015300
227	0.016900
228	0.015600
229	0.016200
230	0.014800

231	0.035400
232	0.008700
233	0.006500
234	0.033600
235	0.020600
236	0.013000
237	0.015700
238	0.035900
239	0.027400
240	0.010200
241	0.034500
242	0.008900
243	0.035700
244	0.013700
245	0.033700
246	0.030000
247	0.004900
248	0.007300
249	0.018300
250	0.022600
251	0.011500
252	0.020700
253	0.011000
254	0.002500
255	0.002600
256	0.043300
257	0.010900
258	0.019100
259	0.010100
260	0.011600
261	0.009800
262	0.005100
263	0.003700
264	0.012700
265	0.011700
266	0.017700
267	0.008700
268	0.014000
269	0.012900
270	0.001800
271	0.015600
272	0.008900
273	0.012300
274	0.009200
275	0.005700
276	0.030500
277	0.018100

278	0.010200
279	0.011000
280	0.019000
281	0.004700
282	0.016700
283	0.019700
284	0.004700
285	0.005700
286	0.012000
287	0.007100
288	0.027400
289	0.002200
290	0.005700
291	0.061200
292	0.018300
293	0.006800
294	0.009400
295	0.002100
296	0.008300
297	0.007800
298	0.009300
299	0.014000
300	0.010500
301	0.002900
302	0.016200
303	0.002800
304	0.008500
305	0.003700
306	0.010900
307	0.005000
308	0.005200
309	0.004400
310	0.003000
311	0.006700
312	0.009100
313	0.008500
314	0.004200
315	0.011900
316	0.010500
317	0.008000
318	0.005600
319	0.007600
320	0.011000

Show final memory and time stats

```
#@title Show final memory and time stats
used_memory = round(torch.cuda.max_memory_reserved() / 1024 / 1024 / 1024, 3)
used_memory_for_lora = round(used_memory - start_gpu_memory, 3)
used_percentage = round(used_memory         /max_memory*100, 3)
lora_percentage = round(used_memory_for_lora/max_memory*100, 3)
print(f"{trainer_stats.metrics['train_runtime']} seconds used for training.")
print(f"{round(trainer_stats.metrics['train_runtime']/60, 2)} minutes used for training.")
print(f"Peak reserved memory = {used_memory} GB.")
print(f"Peak reserved memory for training = {used_memory_for_lora} GB.")
print(f"Peak reserved memory % of max memory = {used_percentage} %.")
print(f"Peak reserved memory for training % of max memory = {lora_percentage} %.")
```

```
336.9222 seconds used for training.
5.62 minutes used for training.
Peak reserved memory = 3.74 GB.
Peak reserved memory for training = 0.0 GB.
Peak reserved memory % of max memory = 25.359 %.
Peak reserved memory for training % of max memory = 0.0 %.
```

Inference

Let's run the model! You can change the instruction and input - leave the output blank!

[NEW] Try 2x faster inference in a free Colab for Llama-3.1 8b Instruct here

We use `min_p = 0.1` and `temperature = 1.5`. Read this Tweet for more information on why.

```
from unsloth.chat_templates import get_chat_template

tokenizer = get_chat_template(
    tokenizer,
    chat_template = "llama-3.1",
)
FastLanguageModel.for_inference(model) # Enable native 2x faster inference

messages = [
    {"role": "user", "content": "Can a student enrolled in MS AI take MS Software Engineering core courses..? "},
]
inputs = tokenizer.apply_chat_template(
    messages,
    tokenize = True,
    add_generation_prompt = True, # Must add for generation
    return_tensors = "pt",
).to("cuda")

outputs = model.generate(input_ids = inputs, max_new_tokens = 64, use_cache = True,
                         temperature = 1.5, min_p = 0.1)
tokenizer.batch_decode(outputs)
```

```
['<|begin_of_text|><|start_header_id|>system<|end_header_id|>\n\nCutting Knowledge Date: December 2023\nToday Date: 26 July
2024\n\n<|eot_id|><|start_header_id|>user<|end_header_id|>\n\nCan a student enrolled in MS AI take MS Software Engineering
core courses..? <|eot_id|><|start_header_id|>assistant<|end_header_id|>\n\nYes, students enrolled in MS AI can take MS
Software Engineering core courses, but they must meet the program requirements and prerequisites.<|eot_id|>']
```

You can also use a `TextStreamer` for continuous inference - so you can see the generation token by token, instead of waiting the whole time!

```
FastLanguageModel.for_inference(model) # Enable native 2x faster inference

messages = [
    {"role": "user", "content": "Can a student enrolled in MS AI take MS Software Engineering core courses..?"},
]
inputs = tokenizer.apply_chat_template(
    messages,
    tokenize = True,
    add_generation_prompt = True, # Must add for generation
    return_tensors = "pt",
).to("cuda")

from transformers import TextStreamer
text_streamer = TextStreamer(tokenizer, skip_prompt = True)
```

```
_ = model.generate(input_ids = inputs, streamer = text_streamer, max_new_tokens = 128,
                   use_cache = True, temperature = 1.5, min_p = 0.1)
```

```
Yes, students can take MS Software Engineering core courses, but they must meet the program requirements and prerequisites.<
```

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.

GGUF / llama.cpp Conversion

To save to `GGUF / llama.cpp`, unsloth support it natively now! cloneed `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.

```
# Save to 8bit Q8_0
if False: 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 True: model.save_pretrained_gguf("llama_32_1b_advisor", tokenizer, quantization_method = "f16")
# if False: model.push_to_hub_gguf("hf/model", tokenizer, quantization_method = "f16", token = "")
if False: model.push_to_hub_gguf("hf/llama_32_1b_advisor", tokenizer, quantization_method = "f16")

# 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
    )


if True: model.save_pretrained_gguf("llama_32_1b_advisor", tokenizer, quantization_method = "f16")
```

```
" }}
        {%- endfor %}
        {{- first_user_message + "<|eot_id|>"}}
    {%- endif %}

    {%- for message in messages %}
        {%- if not (message.role == 'ipython' or message.role == 'tool' or 'tool_calls' in message) %}
            {{- '<|start_header_id|>' + message['role'] + '<|end_header_id|>

    '+ message['content'] + '<|eot_id|>' }}
        {%- elif 'tool_calls' in message %}
            {%- if not message.tool_calls|length == 1 %}
                {{- raise_exception("This model only supports single tool-calls at once!") }}
            {%- endif %}
            {%- set tool_call = message.tool_calls[0].function %}
            {%- if builtin_tools is defined and tool_call.name in builtin_tools %}
                {{- '<|start_header_id|>assistant<|end_header_id|>

    ' -}}
                {{- "<|python_tag|>" + tool_call.name + ".call(" }}
                {%- for arg_name, arg_val in tool_call.arguments | items %}
                    {{- arg_name + '="' + arg_val + '"' }}
                    {%- if not loop.last %}
                        {{- ", " }}
                    {%- endif %}
                    {%- endfor %}
                {{- ")" }}
            {%- else  %}
                {{- '<|start_header_id|>assistant<|end_header_id|>
```

Downloading GGUF file

- downloading gguf file was incredible slow
- Hence, mounted colab on drive, and pushed this gguf file to drive
- downloaded gguf file from drive to local machine
- and Ran it with Ollama

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# give me the command to copy file at /content/llama_32_1b_advisor/unsloth.F16.gguf to /content/drive/MyDrive/SJSU/Semester III/
!cp /content/llama_32_1b_advisor/unsloth.F16.gguf "/content/drive/MyDrive/SJSU/Semester III/CMPE 259: NLP/Project"
```

```
if False: model.push_to_hub_gguf("hf/llama_32_1b_advisor", tokenizer, quantization_method = "f16")
```

Ollama Support and Running it on local machine

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
```

```
>>> Installing ollama to /usr/local
>>> Downloading Linux amd64 bundle
############################################################################################ 100.0%
>>> 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.
```

```
import subprocess
subprocess.Popen(["ollama", "serve"])
```

```
import time


print(tokenizer._ollama_modelfile)
```

```
FROM {__FILE_LOCATION__}
TEMPLATE """{{ if .Messages }}
{{- if or .System .Tools }}<|start_header_id|>system<|end_header_id|>
{{- if .System }}

{{ .System }}
{{- end }}
{{- if .Tools }}

You are a helpful assistant with tool calling capabilities. When you receive a tool call response, use the output to format
{{- end }}
{{- end }}<|eot_id|>
{{- range $i, $_ := .Messages }}
{{- $last := eq (len (slice $.Messages $i)) 1 }}
{{- if eq .Role "user" }}<|start_header_id|>user<|end_header_id|>
{{- if and $.Tools $last }}

Given the following functions, please respond with a JSON for a function call with its proper arguments that best answers th

Respond in the format {"name": function name, "parameters": dictionary of argument name and its value}. Do not use variables

{{ $.Tools }}
{{- end }}

{{ .Content }}<|eot_id|>{{ if $last }}<|start_header_id|>assistant<|end_header_id|>

{{ end }}
{{- else if eq .Role "assistant" }}<|start_header_id|>assistant<|end_header_id|>
{{- if .ToolCalls }}

{{- range .ToolCalls }}{"name": "{{ .Function.Name }}", "parameters": {{ .Function.Arguments }}}{{ end }}
{{- else }}

{{ .Content }}{{ if not $last }}<|eot_id|>{{ end }}
{{- end }}
{{- else if eq .Role "tool" }}<|start_header_id|>ipython<|end_header_id|>

{{ .Content }}<|eot_id|>{{ if $last }}<|start_header_id|>assistant<|end_header_id|>

{{ end }}
{{- end }}
{{- end }}
{{- else }}
{{- if .System }}<|start_header_id|>system<|end_header_id|>

{{ .System }}<|eot_id|>{{ end }}{{ if .Prompt }}<|start_header_id|>user<|end_header_id|>

{{ .Prompt }}<|eot_id|>{{ end }}<|start_header_id|>assistant<|end_header_id|>

{{ end }}{{ .Response }}{{ if .Response }}<|eot_id|>{{ end }}"""
PARAMETER stop "<|start_header_id|>"
PARAMETER stop "<|end_header_id|>"
PARAMETER stop "<|eot_id|>"
PARAMETER stop "<|eom_id|>"
```