```
%%capture
import torch
major_version, minor_version = torch.cuda.get_device_capability()
# Must install separately since Colab has torch 2.2.1, which breaks packages
!pip install "unsloth[colab-new] @ git+https://github.com/unslothai/unsloth.git"
if major_version >= 8:
    # Use this for new GPUs like Ampere, Hopper GPUs (RTX 30xx, RTX 40xx, A100, H100, L40)
    !pip install --no-deps packaging ninja einops flash-attn xformers trl peft accelerate bitsandbytes
else:
    # Use this for older GPUs (V100, Tesla T4, RTX 20xx)
    !pip install --no-deps xformers trl peft accelerate bitsandbytes
pass
```

```
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-bnb-4bit",
#     "unsloth/mistral-7b-instruct-v0.2-bnb-4bit",
#     "unsloth/llama-2-7b-bnb-4bit",
#     "unsloth/gemma-7b-bnb-4bit",
#     "unsloth/gemma-7b-it-bnb-4bit", # Instruct version of Gemma 7b
#     "unsloth/gemma-2b-bnb-4bit",
#     "unsloth/gemma-2b-it-bnb-4bit", # Instruct version of Gemma 2b
#     "unsloth/llama-3-8b-bnb-4bit", # [NEW] 15 Trillion token Llama-3
# ] # 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
)
```

```
config.json: 100%  1.14k/1.14k [00:00<00:00, 11.7kB/s]
==((====))==  Unsloth: Fast Llama patching release 2024.4
   \\   /|    GPU: Tesla T4. Max memory: 14.748 GB. Platform = Linux.
O^O/ \_/ \    Pytorch: 2.2.1+cu121. CUDA = 7.5. CUDA Toolkit = 12.1.
\        /    Bfloat16 = FALSE. Xformers = 0.0.25.post1. FA = False.
 "-____-"     Free Apache license: http://github.com/unslothai/unsloth
Unused kwargs: ['_load_in_4bit', '_load_in_8bit', 'quant_method']. These kwargs are not used in <class 'transformers.ut
model.safetensors: 100%  5.70G/5.70G [00:56<00:00, 130MB/s]
generation_config.json: 100%  131/131 [00:00<00:00, 8.54kB/s]
tokenizer_config.json: 100%  50.6k/50.6k [00:00<00:00, 2.55MB/s]
tokenizer.json: 100%  9.09M/9.09M [00:00<00:00, 26.4MB/s]
special_tokens_map.json: 100%  449/449 [00:00<00:00, 22.7kB/s]
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
```

```python
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 2024.4 patched 32 layers with 32 QKV layers, 32 O layers and 32 MLP layers.
```

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

### Instruction:
{}

### Input:
{}

### Response:
{}"""

EOS_TOKEN = tokenizer.eos_token # Must add EOS_TOKEN
def formatting_prompts_func(examples):
    instructions = examples["instruction"]
    inputs       = examples["input"]
    outputs      = examples["output"]
    texts = []
    for instruction, input, output in zip(instructions, inputs, outputs):
        # Must add EOS_TOKEN, otherwise your generation will go on forever!
        text = alpaca_prompt.format(instruction, input, output) + EOS_TOKEN
        texts.append(text)
    return { "text" : texts, }
pass

from datasets import load_dataset
dataset = load_dataset("Basit4x/HRDataset", split = "train")
dataset = dataset.map(formatting_prompts_func, batched = True,)
```

```
Downloading readme: 100%  73.1k/73.1k [00:00<00:00, 3.68MB/s]
Repo card metadata block was not found. Setting CardData to empty.
WARNING:huggingface_hub.repocard:Repo card metadata block was not found. Setting CardData to empty.
Downloading data: 100%  73.1k/73.1k [00:00<00:00, 1.13MB/s]
Generating train split: 100%  199/199 [00:00<00:00, 4359.85 examples/s]
Map: 100%  199/199 [00:00<00:00, 3899.93 examples/s]
```

```
from trl import SFTTrainer
from transformers import TrainingArguments

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,
        num_train_epochs=2,
        learning_rate = 2e-4,
        fp16 = not torch.cuda.is_bf16_supported(),
        bf16 = torch.cuda.is_bf16_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()
  self.pid = os.fork()
Map (num_proc=2): 100% ████████████████████ 199/199 [00:02<00:00, 83.82 examples/s]
```

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.
5.605 GB of memory reserved.
```

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

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs = 1
   \\   /|    Num examples = 199 | Num Epochs = 2
O^O/ \_/ \    Batch size per device = 2 | Gradient Accumulation steps = 4
\        /    Total batch size = 8 | Total steps = 50
 "-____-"     Number of trainable parameters = 41,943,040
```

[50/50 03:45, Epoch 2/2]

Step	Training Loss
1	3.184200
2	3.085200
3	3.047100
4	2.768300
5	2.586900
6	2.228300
7	2.092400
8	1.700700
9	1.347700
10	1.312700
11	1.241500
12	1.327900
13	1.356600
14	1.352500
15	1.060000
16	1.264100
17	1.238300
18	1.177400
19	1.133800
20	1.180100
21	1.138100
22	0.972500
23	1.192700
24	1.227200
25	1.228900

26	1.087900
27	1.025800
28	0.976200
29	0.932600
30	0.967000
31	1.070900
32	0.971400
33	0.939500
34	1.158100
35	0.982400
36	0.920900
37	0.895800
38	0.926700
39	1.083700
40	1.094500
41	0.940800
42	0.944500
43	0.863000
44	0.843700
45	0.942700
46	0.726300
47	0.881800
48	0.872600
49	0.702000
50	0.873400

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} %.")
```

```
237.6965 seconds used for training.
3.96 minutes used for training.
Peak reserved memory = 7.002 GB.
Peak reserved memory for training = 1.397 GB.
Peak reserved memory % of max memory = 47.478 %.
Peak reserved memory for training % of max memory = 9.472 %.
```

```
# alpaca_prompt = Copied from above
FastLanguageModel.for_inference(model) # Enable native 2x faster inference
inputs = tokenizer(
[
    alpaca_prompt.format(
        "How are employees of Scale I to VII appointed, and who serves as the appointing authority in this case?", # instruction
        "", # input
        "", # output - leave this blank for generation!
    )
], return_tensors = "pt").to("cuda")

outputs = model.generate(**inputs, max_new_tokens = 64, use_cache = True)
tokenizer.batch_decode(outputs)
```

```
Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.
['<|begin_of_text|>Below is an instruction that describes a task, paired with an input that provides further context. Write a response that
appropriately completes the request.\n\n### Instruction:\nHow are employees of Scale I to VII appointed, and who serves as the appointing authority in
this case?\n\n### Input:\n\n\n### Response:\nEmployees of Scale I to VII are appointed by the Rector, who is the appointing authority. The Rector may
delegate this authority to the Campus Director, who may further delegate it to the Campus Director (Reporting). The Campus Director (Reporting) may
further delegate this authority to the Campus Director (Reporting) or the']
```

```
model.save_pretrained("LLM-FAST-Model") # Local saving
# model.push_to_hub("your_name/lora_model", token = "...") # Online saving
```

```
if True: model.save_pretrained_merged("model", tokenizer, save_method = "merged_16bit",)
if True: model.push_to_hub_merged("Methooos/LLM-FAST-Model", tokenizer, save_method = "merged_16bit", token = "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")
```

```
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 6.25 out of 12.67 RAM for saving.
 47%|████▋     | 15/32 [00:01<00:01, 15.51it/s]We will save to Disk and not RAM now.
100%|██████████| 32/32 [00:53<00:00,  1.69s/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: You are pushing to hub, but you passed your HF username = Methooos.
We shall truncate Methooos/LLM-FAST-Model to LLM-FAST-Model
Unsloth: Merging 4bit and LoRA weights to 16bit...
Unsloth: Will use up to 5.88 out of 12.67 RAM for saving.
```