```
%%capture
import torch
major_version, minor_version = torch.cuda.get_device_capability()
!pip install "unsloth[colab-new] @ git+https://github.com/unslothai/unsloth.git"
if major_version >= 8:
    !pip install --no-deps packaging ninja einops flash-attn xformers trl peft accelerate bitsandbytes
else:
    !pip install --no-deps xformers trl peft accelerate bitsandbytes
pass
```

```
from unsloth import FastLanguageModel
import torch
max_seq_length = 2048
dtype = None
load_in_4bit = True

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",
    "unsloth/gemma-2b-bnb-4bit",
    "unsloth/gemma-2b-it-bnb-4bit",
    "unsloth/llama-3-8b-bnb-4bit",
]

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,
)
```

```
🦥 Unsloth: Will patch your computer to enable 2x faster free finetuning.
==((====))==  Unsloth: Fast Llama patching release 2024.7
   \\   /|    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. FA [Xformers = 0.0.26.post1. FA2 = False]
 "-____-"     Free Apache license: http://github.com/unslothai/unsloth
```

```
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,
    bias = "none",
    use_gradient_checkpointing = "unsloth",
    random_state = 3407,
    use_rslora = False,
    loftq_config = None,
)
```

```
Unsloth 2024.7 patched 32 layers with 32 QKV layers, 32 O layers and 32 MLP layers.
```

```python
# this is basically the system prompt
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:
{}

### output:
{}"""

EOS_TOKEN = tokenizer.eos_token # do not forget this part!
def formatting_prompts_func(examples):
    instructions = examples["instruction"]
    inputs       = examples["input"]
    outputs      = examples["output"]
    texts = []
    for instruction, input, output in zip(instructions, inputs, outputs):
        text = alpaca_prompt.format(instruction, input, output) + EOS_TOKEN # without this token generation goes on forever!
        texts.append(text)
    return { "text" : texts, }
pass

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

```
Map: 100% 20000/20000 [00:00<00:00, 27228.94 examples/s]
```

```python
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,
        max_steps = 180, # increase this to make the model learn "better"
        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/transformers/training_args.py:1961: FutureWarning: `--push_to_hub_token` is deprecated and will be removed in version 5 of 🤗 Transformers. Use `--hub_token` instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/trl/trainer/sft_trainer.py:269: UserWarning: You passed a `max_seq_length` argument to the SFTTrainer, the value you passed will override the one in the `SFTConfig`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/trl/trainer/sft_trainer.py:283: UserWarning: You passed a `dataset_num_proc` argument to the SFTTrainer, the value you passed will override the one in the `SFTConfig`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/trl/trainer/sft_trainer.py:307: UserWarning: You passed a `dataset_text_field` argument to the SFTTrainer, the value you passed will override the one in the `SFTConfig`.
  warnings.warn(
max_steps is given, it will override any value given in num_train_epochs
```

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

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs = 1
   \\   /|    Num examples = 20,000 | Num Epochs = 1
O^O/ \_/ \    Batch size per device = 2 | Gradient Accumulation steps = 4
\        /    Total batch size = 8 | Total steps = 180
 "-____-"     Number of trainable parameters = 41,943,040
```

[180/180 22:44, Epoch 0/1]

Step	Training Loss
1	2.933100
2	2.861600
3	2.810800
4	2.718400
5	2.613100
6	2.270000
7	2.053200
8	1.816700
9	1.726100
10	1.348200
11	1.098200
12	1.176100
13	1.201200
14	1.067200
15	1.216500
16	1.079300
17	1.063200

18	1.212300
19	1.130200
20	1.240900
21	1.152800
22	1.004700
23	1.048400
24	0.987600
25	1.119200
26	0.994900
27	1.117900
28	0.953900
29	1.095600
30	1.172400
31	1.093200
32	0.866500
33	1.000500
34	0.957400
35	1.023000
36	1.126900
37	0.895800
38	0.957400
39	0.904700

40	1.131500
41	0.848500
42	1.228500
43	0.983200
44	1.060800
45	0.989500
46	1.152500
47	0.971800
48	1.037800
49	0.865100
50	0.972600
51	1.052600
52	0.881300
53	1.118300
54	0.998300
55	1.043900
56	1.090600
57	0.895500
58	1.051700
59	0.913700
60	1.060600
61	1.031700

61	1.031700
62	1.061400
63	0.905500
64	1.047300
65	0.998400
66	0.874000
67	0.897900
68	0.943800
69	1.012300
70	0.888900
71	1.125100
72	0.979500
73	0.970200
74	1.031400
75	0.981400
76	0.811900
77	1.108900
78	0.953800
79	0.971800
80	0.980600
81	1.124900
82	1.152700
83	0.926100
84	1.039700

85	0.925800
86	0.773700
87	1.149200
88	1.030900
89	1.020200
90	0.932800
91	0.998800
92	0.969400
93	0.964700
94	0.941000
95	1.026000
96	1.184800
97	0.941200
98	0.941300
99	1.125100
100	1.083400
101	1.045400
102	1.026000
103	1.060500
104	1.009700
105	0.928900
106	1.013000
107	1.008800

108	0.968500
109	0.952400
110	1.102000
111	1.078000
112	0.971800
113	0.979400
114	0.985600
115	0.950800
116	0.931200
117	1.023400
118	1.010600
119	0.941300
120	1.035300
121	0.958100
122	1.165400
123	0.966300
124	0.853100
125	1.077300
126	0.977500
127	0.914600
128	0.923700
129	0.939600
130	0.984400

131	0.868800
132	1.117700
133	0.943100
134	1.103100
135	0.876800
136	1.040400
137	0.919000
138	1.086200
139	0.890200
140	1.018700
141	0.968500
142	1.182400
143	1.164400
144	0.997100
145	0.824200
146	0.866800
147	1.017700
148	0.957600
149	1.023000
150	0.846900
151	0.970900
152	1.023800
153	0.941700
154	0.960800

154	0.960800
155	0.981400
156	0.939900
157	0.882800
158	0.906800
159	0.935900
160	0.914500
161	0.953700
162	0.986900
163	1.064600
164	1.008100
165	1.038000
166	0.920700
167	0.946900
168	0.943200
169	0.979800
170	1.065200
171	1.039900
172	0.887800
173	0.861900
174	0.996300
175	0.959200
176	0.933200
177	1.150000

178	1.208800
179	0.868100
180	0.905200

```
FastLanguageModel.for_inference(model)
inputs = tokenizer(
[
    alpaca_prompt.format(
        "You are a professional chef assistant. Make a dish by firstly presenting it, then listing the ingredients and finally the recipe step by step in bullet points under the following input requirements.", # instruction
        "Requires 40 minutes preparation or less and it contains at least this ingredients mayonnaise, green chilies, sharp cheddar cheese", # input
        "", # output - leave this blank for generation!
    )
], return_tensors = "pt").to("cuda")

outputs = model.generate(**inputs, max_new_tokens = 128, 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:\nYou are a professional chef
assistant. Make a dish by firstly presenting it, then listing the ingredients and finally the recipe step by step in bullet points under the following input requirements.\n\n### input:\nRequires 40 minutes preparation or less and it
contains at least this ingredients mayonnaise, green chilies, sharp cheddar cheese\n\n### output:\nThis is a recipe i found in a cookbook that is based on the southwest.  it is a very easy to make dip and it is great for parties.  i
have also used it as a spread on sandwiches.  it is delicious.\nYou will need this ingredients: mayonnaise, sour cream, sharp cheddar cheese, green chilies, onion, salt and pepper, garlic powder\n\nSteps:\n• Mix all ingredients
together in a bowl\n• Cover and refrigerate for at least 2 hours\n• Serve with chips or crackers\n<|end_of_text|>']
```

```
FastLanguageModel.for_inference(model)
inputs = tokenizer(
[
    alpaca_prompt.format(
        "You are a professional chef assistant. Make a dish by firstly presenting it, then listing the ingredients and finally the recipe step by step in bullet points under the following input requirements.", # instruction
        "Requires 40 minutes preparation or less and it contains at least this ingredients chicken", # input
        "", # output - leave this blank for generation!
    )
], return_tensors = "pt").to("cuda")

from transformers import TextStreamer
text_streamer = TextStreamer(tokenizer)
_ = model.generate(**inputs, streamer = text_streamer, max_new_tokens = 128)
```

```
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.

### instruction:
You are a professional chef assistant. Make a dish by firstly presenting it, then listing the ingredients and finally the recipe step by step in bullet points under the following input requirements.

### input:
Requires 40 minutes preparation or less and it contains at least this ingredients chicken

### output:
This is a great dish for the summer.  you can use any chicken you have on hand.  i used boneless chicken breasts and boneless chicken thighs.  it is a great way to use up leftover chicken or rotisserie chicken.  it can be served hot
You will need this ingredients: chicken, green beans, red onion, celery, carrots, red wine vinegar, olive oil, salt, pepper, basil

Steps:
• In a large bowl, combine all ingredients
• Chill at least 4 hours before serving
• Serve cold or warm
<|end_of_text|>
```

```
from huggingface_hub import notebook_login
notebook_login()
```

Token is valid (permission: write).

Your token has been saved in your configured git credential helpers (store).

Your token has been saved to /root/.cache/huggingface/token

Login successful

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

```
model.save_pretrained("lora_model") # Local saving

model.push_to_hub("bsc-9/AI_Cooking_Kitchen", token = "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") # Online saving
```

```
README.md:  0.00/0.00 [00:00<?, ?B/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.
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.
adapter_model.safetensors: 100% 168M/168M [00:18<00:00, 13.5MB/s]
Saved model to https://huggingface.co/bsc-9/AI_Cooking_Kitchen
```