Start coding or generate with AI.

```
# Example: Baseline inference with a 1B instruct model (zero-shot)
!pip install unsloth transformers datasets -q
from datasets import load_dataset
from unsloth import FastLanguageModel
import torch

# Load data (take 75%-agreement subset) and split
dataset = load_dataset("takala/financial_phrasebank", "sentences_75agree")
data = dataset["train"]
data = data.train_test_split(test_size=0.2, seed=42)
train_data = data["train"]
test_data = data["test"]

# Load base model in 4-bit (QLoRA) for speed
model_name = "unsloth/Llama-3.2-1B-Instruct-bnb-4bit"
model, tokenizer = FastLanguageModel.from_pretrained(model_name, max_seq_length=2048, load_in_4bit=True)
FastLanguageModel.for_inference(model)  # prepare for faster generation

# Example inference on one sample
sample = test_data[0]["sentence"]
prompt = f"{sample} Sentiment:"
inputs = tokenizer(prompt, return_tensors="pt").to("cuda")
outs = model.generate(input_ids=inputs['input_ids'], max_new_tokens=3, temperature=0.2)
label = tokenizer.decode(outs[0], skip_special_tokens=True)
print("Sentence:", sample)
print("Predicted sentiment:", label)
```

```
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 52.3/52.3 kB 3.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 311.7/311.7 kB 13.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 491.5/491.5 kB 30.2 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 511.9/511.9 kB 44.9 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 184.8/184.8 kB 18.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 117.2/117.2 MB 8.0 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.3/61.3 MB 13.4 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 130.0/130.0 kB 12.0 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 213.6/213.6 kB 20.2 MB/s eta 0:00:00
🦥 Unsloth: Will patch your computer to enable 2x faster free finetuning.
🦥 Unsloth Zoo will now patch everything to make training faster!
README.md: 8.88k/? [00:00<00:00, 711kB/s]
financial_phrasebank.py: 6.04k/? [00:00<00:00, 578kB/s]
The repository for takala/financial_phrasebank contains custom code which must be executed to correctly load the dataset. You can inspec
You can avoid this prompt in future by passing the argument `trust_remote_code=True`.

Do you wish to run the custom code? [y/N] y
FinancialPhraseBank-v1.0.zip: 100%  682k/682k [00:00<00:00, 8.38MB/s]
Generating train split: 100%  3453/3453 [00:00<00:00, 42658.84 examples/s]
==((====))==  Unsloth 2025.8.9: Fast Llama patching. Transformers: 4.55.4.
   \\   /|    Tesla T4. Num GPUs = 1. Max memory: 14.741 GB. Platform: Linux.
O^O/ \_/ \    Torch: 2.8.0+cu126. CUDA: 7.5. CUDA Toolkit: 12.6. Triton: 3.4.0
\        /    Bfloat16 = FALSE. FA [Xformers = 0.0.32.post2. FA2 = False]
 "-____-"     Free 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 [00:08<00:00, 221MB/s]
generation_config.json: 100%  234/234 [00:00<00:00, 14.4kB/s]
tokenizer_config.json: 54.7k/? [00:00<00:00, 5.27MB/s]
tokenizer.json: 100%  17.2M/17.2M [00:00<00:00, 49.1MB/s]
special_tokens_map.json: 100%  454/454 [00:00<00:00, 37.5kB/s]
The attention mask is not set and cannot be inferred from input because pad token is same as eos token. As a consequence, you may observ
Sentence: It estimates the operating profit to further improve from the third quarter .
Predicted sentiment: It estimates the operating profit to further improve from the third quarter. Sentiment: positive. The
```

```
print(test_data[3])
```

```
{'sentence': 'Our tools are specifically designed with the needs of both the business users and ICT experts in mind .', 'label': 1}
```

```
!pip install scikit-learn -q
from sklearn.metrics import accuracy_score, f1_score

# Label mapping
label_map = {"negative": 0, "neutral": 1, "positive": 2}

def normalize_pred(text):
    text = text.lower()
    if "pos" in text:
        return 2   # positive
    elif "neg" in text:
        return 0   # negative
    elif "neu" in text:
        return 1   # neutral
    else:
        return 1   # fallback to neutral

y_true = []
y_pred = []

for sample in test_data:
    sent = sample["sentence"]
    gold = sample["label"]  # already int (0/1/2)

    prompt = f"Sentence: {sent}\nSentiment:"
    inputs = tokenizer(prompt, return_tensors="pt").to("cuda")

    outputs = model.generate(
        input_ids=inputs["input_ids"],
        attention_mask=inputs["attention_mask"],
        max_new_tokens=3,
        temperature=0.2,
    )
    pred_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
    pred = normalize_pred(pred_text)

    y_true.append(gold)
    y_pred.append(pred)

# Compute metrics
acc = accuracy_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred, average="macro")
print(f"Baseline Accuracy: {acc*100:.2f}%")
print(f"Baseline F1 (macro): {f1*100:.2f}%")
```

```
Baseline Accuracy: 61.79%
Baseline F1 (macro): 30.51%
```

```
from unsloth import FastLanguageModel

# Load base model (16-bit for LoRA training)
model, tokenizer = FastLanguageModel.from_pretrained(
    "unsloth/Llama-3.2-1B",
    max_seq_length=2048,
    load_in_4bit=False,
)

# Wrap with LoRA
model = FastLanguageModel.get_peft_model(
    model,
    r=16,
    target_modules=["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","down_proj"],
    lora_alpha=16,
    lora_dropout=0.0,
    bias="none",
    use_gradient_checkpointing="unsloth",
)

model.print_trainable_parameters()
```

```
==((====))==  Unsloth 2025.8.9: Fast Llama patching. Transformers: 4.55.4.
   \\   /|    Tesla T4. Num GPUs = 1. Max memory: 14.741 GB. Platform: Linux.
O^O/ \_/ \    Torch: 2.8.0+cu126. CUDA: 7.5. CUDA Toolkit: 12.6. Triton: 3.4.0
\        /    Bfloat16 = FALSE. FA [Xformers = 0.0.32.post2. FA2 = False]
 "-____-"     Free license: http://github.com/unslothai/unsloth
Unsloth: Fast downloading is enabled - ignore downloading bars which are red colored!
```

model.safetensors: 100% 2.47G/2.47G [00:32<00:00, 262MB/s]

generation_config.json: 100% 230/230 [00:00<00:00, 15.0kB/s]

tokenizer_config.json: 50.6k/? [00:00<00:00, 4.92MB/s]

special_tokens_map.json: 100% 459/459 [00:00<00:00, 52.5kB/s]

tokenizer.json: 100% 17.2M/17.2M [00:00<00:00, 31.5MB/s]

```
Unsloth 2025.8.9 patched 16 layers with 16 QKV layers, 16 O layers and 16 MLP layers.
trainable params: 11,272,192 || all params: 1,247,086,592 || trainable%: 0.9039
```

```python
id2label = {0: "negative", 1: "neutral", 2: "positive"}

def formatting_func(example):
    texts = []
    sentences = example["sentence"]
    labels = example["label"]

    # Case 1: batch (lists)
    if isinstance(sentences, list):
        for sent, lab in zip(sentences, labels):
            texts.append(f"Sentence: {sent}\nSentiment: {id2label[lab]}")
    else:
        # Case 2: single sample
        texts.append(f"Sentence: {sentences}\nSentiment: {id2label[labels]}")
    return texts
```

```python
from transformers import TrainingArguments
from unsloth.trainer import SFTTrainer

args = TrainingArguments(
    output_dir="outputs_lora_1b",
    num_train_epochs=3,
    per_device_train_batch_size=2,
    gradient_accumulation_steps=4,
    learning_rate=2e-4,
    fp16=True,
    logging_steps=50,
    save_strategy="no",
)

trainer = SFTTrainer(
    model=model,
    tokenizer=tokenizer,
    train_dataset=train_data,
    eval_dataset=test_data,
    formatting_func=formatting_func,
    max_seq_length=512,
    args=args,
)
```

```python
trainer.train()
```

Unsloth: Tokenizing ["text"]: 100% 2762/2762 [00:01<00:00, 2435.79 examples/s]

Unsloth: Tokenizing ["text"]: 100% 691/691 [00:00<00:00, 2955.41 examples/s]

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs used = 1
   \\   /|    Num examples = 2,762 | Num Epochs = 3 | Total steps = 1,038
O^O/ \_/ \    Batch size per device = 2 | Gradient accumulation steps = 4
\        /    Data Parallel GPUs = 1 | Total batch size (2 x 4 x 1) = 8
 "-____-"     Trainable parameters = 11,272,192 of 1,247,086,592 (0.90% trained)
/usr/local/lib/python3.12/dist-packages/notebook/notebookapp.py:191: SyntaxWarning: invalid escape sequence '\/'
  | |_| | '_ \/ _` / _` |  _/ -_)
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize?ref=models
wandb: Paste an API key from your profile and hit enter:wandb: WARNING If you're specifying your api key in code, ensure this code is
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: aravindyuvraj007 (aravindyuvraj007-vnrvjietofficial) to https://api.wandb.ai. Use `wandb login --relogi
```

Tracking run with wandb version 0.21.1

Run data is saved locally in /content/wandb/run-20250828_112757-19xaw7pe

Syncing run **easy-universe-2** to Weights & Biases (docs)

View project at https://wandb.ai/aravindyuvraj007-vnrvjietofficial/huggingface

View run at https://wandb.ai/aravindyuvraj007-vnrvjietofficial/huggingface/runs/19xaw7pe

[362/1038 04:37 < 08:41, 1.30 it/s, Epoch 1.04/3]

Step	Training Loss
50	2.756500
100	2.333300
150	2.237900
200	2.260500
250	2.180100
300	2.213300
350	2.137000

[1038/1038 13:16, Epoch 3/3]

Step	Training Loss
50	2.756500
100	2.333300
150	2.237900
200	2.260500
250	2.180100
300	2.213300
350	2.137000
400	1.888300
450	1.904700

```
from sklearn.metrics import accuracy_score, f1_score

y_true, y_pred = [], []

id2label = {0: "negative", 1: "neutral", 2: "positive"}
label2id = {v:k for k,v in id2label.items()}

for ex in test_data:
    prompt = f"Sentence: {ex['sentence']}\nSentiment:"
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
    outputs = model.generate(**inputs, max_new_tokens=5)
    pred = tokenizer.decode(outputs[0], skip_special_tokens=True)

    # extract last token as sentiment guess
    guess = None
    for cand in id2label.values():
        if cand in pred.lower():
            guess = cand
            break
    if guess is None:
        guess = "neutral"   # fallback

    y_true.append(ex["label"])
```

```
    y_pred.append(label2id[guess])

# Compute metrics
acc = accuracy_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred, average="macro")
print(f"Fine-tuned LoRA Accuracy: {acc*100:.2f}%")
print(f"Fine-tuned LoRA F1 (macro): {f1*100:.2f}%")
```

```
Fine-tuned LoRA Accuracy: 90.74%
Fine-tuned LoRA F1 (macro): 89.05%
```

Start coding or generate with AI.

```
# Test the fine-tuned model with a new prompt
new_prompt = "Sentence: I am not feeling good, I am feeling very not good and not good heart attack.\nSentiment:"
inputs = tokenizer(new_prompt, return_tensors="pt").to(model.device)

outputs = model.generate(**inputs, max_new_tokens=5)
generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)

print("Generated Sentiment:", generated_text)
```

```
Generated Sentiment: Sentence: I am not feeling good, I am feeling very not good and not good heart attack.
Sentiment: negative - negative Sentiment
```

Start coding or generate with AI.

Task

Set up the code to fine-tune a language model using QLoRA with the `FastLanguageModel` library. Load the model in 4-bit precision, wrap it with LoRA, define training arguments, and initialize the `SFTTrainer`. Do not run the training.

Load the model in 4-bit

Subtask:

Load the base model in 4-bit precision using `FastLanguageModel.from_pretrained`.

Reasoning: The subtask is to load the base model in 4-bit precision. This requires using the `FastLanguageModel.from_pretrained` method with the specified parameters.

```
from unsloth import FastLanguageModel

# Load base model in 4-bit for QLoRA fine-tuning
model, tokenizer = FastLanguageModel.from_pretrained(
    "unsloth/Llama-3.2-3B-bnb-4bit",
    max_seq_length=2048,
    load_in_4bit=True,
)

peft_model = FastLanguageModel.get_peft_model(
    model,
    r=16,
    target_modules=["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","down_proj"],
    lora_alpha=16,
    lora_dropout=0.0,
    bias="none",
    use_gradient_checkpointing="unsloth"
)

peft_model.print_trainable_parameters()
```

```
==((====))==  Unsloth 2025.8.9: Fast Llama patching. Transformers: 4.55.4.
   \\   /|    Tesla T4. Num GPUs = 1. Max memory: 14.741 GB. Platform: Linux.
O^O/ \_/ \    Torch: 2.8.0+cu126. CUDA: 7.5. CUDA Toolkit: 12.6. Triton: 3.4.0
\        /    Bfloat16 = FALSE. FA [Xformers = 0.0.32.post2. FA2 = False]
 "-____-"     Free license: http://github.com/unslothai/unsloth
```

```
Unsloth: Fast downloading is enabled - ignore downloading bars which are red colored!
Unsloth 2025.8.9 patched 28 layers with 28 QKV layers, 28 O layers and 28 MLP layers.
trainable params: 24,313,856 || all params: 3,237,063,680 || trainable%: 0.7511
```

```
from transformers import TrainingArguments
from unsloth.trainer import SFTTrainer

args = TrainingArguments(
    output_dir="outputs_lora_1b",
    num_train_epochs=3,
    per_device_train_batch_size=2,
    gradient_accumulation_steps=4,
    learning_rate=2e-4,
    fp16=True,
    logging_steps=50,
    save_strategy="no",
)

trainer = SFTTrainer(
    model=peft_model,
    tokenizer=tokenizer,
    train_dataset=train_data,
    eval_dataset=test_data,
    formatting_func=formatting_func,
    max_seq_length=512,
    args=args,
)
```

```
trainer.train()
```

Unsloth: Tokenizing ["text"]: 100% 2762/2762 [00:00<00:00, 5458.54 examples/s]

Unsloth: Tokenizing ["text"]: 100% 691/691 [00:00<00:00, 4953.17 examples/s]

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs used = 1
   \\   /|    Num examples = 2,762 | Num Epochs = 3 | Total steps = 1,038
O^O/ \_/ \    Batch size per device = 2 | Gradient accumulation steps = 4
\        /    Data Parallel GPUs = 1 | Total batch size (2 x 4 x 1) = 8
 "-____-"     Trainable parameters = 24,313,856 of 3,237,063,680 (0.75% trained)
```

[1038/1038 29:14, Epoch 3/3]

Step	Training Loss
50	2.503700
100	2.219600
150	2.118500
200	2.134100
250	2.057000
300	2.091300
350	2.022600
400	1.771900
450	1.780200
500	1.790400
550	1.720600
600	1.777300
650	1.723300
700	1.671700
750	1.438500
800	1.431600
850	1.447000
900	1.402600
950	1.383300
1000	1.385300

```
TrainOutput(global_step=1038, training_loss=1.7802981264559061, metrics={'train_runtime': 1759.1777, 'train_samples_per_second': 4.71,
'train_steps_per_second': 0.59, 'total_flos': 6197031974510592.0, 'train_loss': 1.7802981264559061})
```

```
from sklearn.metrics import accuracy_score, f1_score

y_true, y_pred = [], []

id2label = {0: "negative", 1: "neutral", 2: "positive"}
label2id = {v:k for k,v in id2label.items()}

for ex in test_data:
    prompt = f"Sentence: {ex['sentence']}\nSentiment:"
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
    outputs = peft_model.generate(**inputs, max_new_tokens=5)
    pred = tokenizer.decode(outputs[0], skip_special_tokens=True)

    # extract last token as sentiment guess
    guess = None
    for cand in id2label.values():
        if cand in pred.lower():
            guess = cand
            break
    if guess is None:
        guess = "neutral"   # fallback

    y_true.append(ex["label"])
    y_pred.append(label2id[guess])

# Compute metrics
acc = accuracy_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred, average="macro")
print(f"Fine-tuned LoRA Accuracy: {acc*100:.2f}%")
```

```
print(f"Fine-tuned LoRA F1 (macro): {f1*100:.2f}%")
```

```
Fine-tuned LoRA Accuracy: 95.66%
Fine-tuned LoRA F1 (macro): 94.98%
```

```
from peft import PromptTuningConfig, get_peft_model, TaskType, PromptTuningInit
```