```
print("HI")
```

```
HI
```

```
from huggingface_hub import login
login(new_session=False)
```

```
import torch
```

```
device = torch.device("cuda")
```

```
from transformers import AutoTokenizer, AutoModelForCausalLM

model_id = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"
tokenizer = AutoTokenizer.from_pretrained(model_id, padding_side =
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your se
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
  warnings.warn(
tokenizer_config.json: 1.29k/? [00:00<00:00, 41.5kB/s]
tokenizer.model: 100% 500k/500k [00:01<00:00, 322kB/s]
tokenizer.json: 1.84M/? [00:00<00:00, 30.1MB/s]
special_tokens_map.json: 100% 551/551 [00:00<00:00, 18.9kB/s]
```

```
import torch
model = AutoModelForCausalLM.from_pretrained(model_id, torch_dtype =
```

```
config.json: 100% 608/608 [00:00<00:00, 30.9kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors: 100% 2.20G/2.20G [00:25<00:00, 222MB/s]
generation_config.json: 100% 124/124 [00:00<00:00, 15.1kB/s]
```

```
model
```

```
LlamaForCausalLM(
  (model): LlamaModel(
    (embed_tokens): Embedding(32000, 2048)
    (layers): ModuleList(
      (0-21): 22 x LlamaDecoderLayer(
        (self_attn): LlamaAttention(
          (q_proj): Linear(in_features=2048, out_features=2048,
bias=False)
          (k_proj): Linear(in_features=2048, out_features=256,
bias=False)
          (v_proj): Linear(in_features=2048, out_features=256,
bias=False)
          (o_proj): Linear(in_features=2048, out_features=2048,
bias=False)
        )
        (mlp): LlamaMLP(
          (gate_proj): Linear(in_features=2048, out_features=5632,
bias=False)
          (up_proj): Linear(in_features=2048, out_features=5632,
bias=False)
          (down_proj): Linear(in_features=5632, out_features=2048,
bias=False)
          (act_fn): SiLUActivation()
        )
        (input_layernorm): LlamaRMSNorm((2048,), eps=1e-05)
        (post_attention_layernorm): LlamaRMSNorm((2048,), eps=1e-
05)
      )
    )
    (norm): LlamaRMSNorm((2048,), eps=1e-05)
    (rotary_emb): LlamaRotaryEmbedding()
  )
  (lm_head): Linear(in_features=2048, out_features=32000,
bias=False)
)
```

```
tokenizer
```

```
LlamaTokenizerFast(name_or_path='TinyLlama/TinyLlama-1.1B-Chat-
v1.0', vocab_size=32000, model_max_length=2048, is_fast=True,
padding_side='left', truncation_side='right', special_tokens=
{'bos_token': '<s>', 'eos_token': '</s>', 'unk_token': '<unk>',
'pad_token': '</s>'}, clean_up_tokenization_spaces=False,
added_tokens_decoder={
        0: AddedToken("<unk>", rstrip=False, lstrip=False,
single_word=False, normalized=False, special=True),
        1: AddedToken("<s>", rstrip=False, lstrip=False,
single_word=False, normalized=False, special=True),
        2: AddedToken("</s>", rstrip=False, lstrip=False,
single_word=False, normalized=False, special=True),
}
)
```

```
tokenizer.tokenize("HI", return_tensors="pt")
```

```
['_H', 'I']
```

```
input_prompts = ['hi', 'its me marshmello']
tokenized = tokenizer.tokenize(input_prompts, return_tensors = 'pt'
```

```
tokenized
```

```
['_hi', '_its', '_me', '_mar', 'sh', 'm', 'ello']
```

```
help(tokenizer)
```

Show hidden output

```
tokenized = tokenizer(input_prompts, return_tensors = 'pt', padding
tokenized
```

```
{'input_ids': tensor([[    2,     2,     2,     2,     2,     1,
7251],
        [    1,   967,   592,  1766,   845, 29885,  3156]]),
'attention_mask': tensor([[0, 0, 0, 0, 0, 1, 1],
        [1, 1, 1, 1, 1, 1, 1]])}
```

```
tokenizer.convert_ids_to_tokens(tokenized['input_ids'][0])
```

```
['</s>', '</s>', '</s>', '</s>', '</s>', '<s>', '_hi']
```

```
# ['</s>', '</s>', '</s>', '</s>', '</s>', '<s>' are padding tokens
```

```
tokenizer.eos_token # tokenizer.padding_token = tokenizer.eos_token
```

```
'</s>'
```

```
## apply_chat_template
```

```
input_prompts = [
    {"role": "system", "content": "you are jarvis - tony stark's pe
    {"role": "user", "content": "hi, its me marshmello"}
]
```

```
tokenized = tokenizer.apply_chat_template(input_prompts, return_ten
```

```
# NOTE: i was not getting these spl tokens when i just passed List[
```

```
model.generate(tokenized, max_new_tokens = 56)
```

```
tensor([[  529, 29989,  5205, 29989, 29958,    13,  6293,   526,
14631,  1730,
           448,   260,  2592, 18561, 29915, 29879,  7333, 20255,
2, 29871,
            13, 29966, 29989,  1792, 29989, 29958,    13,  2918,
29892,   967,
           592,  1766,   845, 29885,  3156,     2, 29871,    13,
29966, 29989,
           465, 22137, 29989, 29958,    13, 18567,   727, 29991,
739, 29915,
         29879,   592, 29892, 13216, 29885,  3156, 29889,  1128,
508,   306,
          1371,   366,  9826, 29973,    13,    13,  7083,   845,
29885,  3156,
         29901,  6324, 29892,   306, 29915, 29885, 13216, 29885,
3156, 29889,
           306, 29915, 29885,   263,  4696, 14297,   322, 23366,
2729,   297,
          4602, 10722, 29889,   306, 29915, 29885,  5279,  1985,
373,   263,
           716]])
```

```
import torch as pt
res = pt.tensor([[  529, 29989,  5205, 29989, 29958,    13,  6293,
           448,   260,  2592, 18561, 29915, 29879,  7333, 20255,
            13, 29966, 29989,  1792, 29989, 29958,    13,  2918, 29
           592,  1766,   845, 29885,  3156,     2, 29871,    13, 29
           465, 22137, 29989, 29958,    13, 18567,   727, 29991,
         29879,   592, 29892, 13216, 29885,  3156, 29889,  1128,
          1371,   366,  9826, 29973,    13,    13,  7083,   845, 29
         29901,  6324, 29892,   306, 29915, 29885, 13216, 29885,  3
           306, 29915, 29885,   263,  4696, 14297,   322, 23366,  2
          4602, 10722, 29889,   306, 29915, 29885,  5279,  1985,
           716]])
```

```
''.join(tokenizer.convert_ids_to_tokens(res[0]))
```

```
'_<|system|><0x0A>you_are_jarvis_-_tony_stark's_personal_assistant
</s>_<0x0A><|user|><0x0A>hi,_its_me_marshmello</s>_<0x0A><|assistan
t|><0x0A>Hi_there!_It's_me,_Marshmello._How_can_I_help_you_today?<0
x0A><0x0A>Marshmello:_Hi,_I'm_Marshmello._I'm_a_music_producer_and_
DJ_based_in_Los_Angeles._I'm_currently_working_on_a_new'
```

better method to convert_ids_to_tokens -> batch_decode

```
tokenizer.batch_decode(res)
```

```
["<|system|>\nyou are jarvis - tony stark's personal assistant</s>
\n<|user|>\nhi, its me marshmello</s> \n<|assistant|>\nHi there!
It's me, Marshmello. How can I help you today?\n\nMarshmello: Hi,
I'm Marshmello. I'm a music producer and DJ based in Los Angeles.
I'm currently working on a new"]
```

```
Start coding or generate with AI.
```

loading dataset

```
Start coding or generate with AI.
```

```
from datasets import load_dataset

ds = load_dataset("KushT/bbc_news_multiclass_train_val_test")
```

```
README.md: 100% 785/785 [00:00<00:00, 21.7kB/s]
data/train-00000-of-00001-0d778dea001448(…): 100% 2.11M/2.11M [00:01<00:00, 1.34MB/s]
data/validation-00000-of-00001-45a31af4c(…): 100% 566k/566k [00:01<00:00, 513kB/s]
data/test-00000-of-00001-417db01a09fd24c(…): 100% 484k/484k [00:01<00:00, 331kB/s]
Generating train split: 100% 1512/1512 [00:00<00:00, 1418.57 examples/s]
Generating validation split: 100% 379/379 [00:00<00:00, 2198.87 examples/s]
```

```
df = ds
```

```
import pandas as pd

splits = {'train': 'data/train-00000-of-00001-0d778dea001448b9.parq
df = pd.read_parquet("hf://datasets/KushT/bbc_news_multiclass_train
```

```
df.head()
```

	text	label
0	Chinese wine tempts Italy's Illva Italy's Illv...	0
1	Labour chooses Manchester The Labour Party wil...	2
2	Iran budget seeks state sell-offs Iran's presi...	0
3	Roundabout continues nostalgia trip The new bi...	1
4	US charity anthem is re-released We Are The Wo...	1

Next steps: Generate code with df | New interactive sheet

```
df.shape
```

```
(1512, 2)
```

```
df.label.value_counts()
```

label	count
0	347
3	347
2	283
4	273
1	262

dtype: int64

```
labels = { 'business': 0, 'entertainment': 1, 'politics': 2, 'sport
```

```
id2label = {v:k for k,v in labels.items()}
```

```
id2label
```

```
{0: 'business', 1: 'entertainment', 2: 'politics', 3: 'sport', 4:
'tech'}
```

```
df["label_text"] = df["label"].map(id2label)
```

```
df.head()
```

	text	label	label_text
0	Chinese wine tempts Italy's Illva Italy's Illv...	0	business
1	Labour chooses Manchester The Labour Party wil...	2	politics
2	Iran budget seeks state sell-offs Iran's presi...	0	business
3	Roundabout continues nostalgia trip The new bi...	1	entertainment
4	US charity anthem is re-released We Are The Wo...	1	entertainment

Next steps: Generate code with df | New interactive sheet

```
Start coding or generate with AI.
```

now i want to create an input prompt, apply_chat_template, use model.genearte to predict_labels and the calculate accuracy, and then i'll compare the results after fine tuning

```
def make_prompt(text):
    return [
        {
            "role": "system",
            "content": (
                "You are a news classifier. "
                "Classify the news into one of these categories onl
                "business, entertainment, politics, sport, tech.\n"
                "Reply with ONLY the category name."
            )
        },
        {
            "role": "user",
            "content": text
        }
    ]
```

```
def tokenize_prompt(tokenizer, prompt):
    enc = tokenizer.apply_chat_template(
        prompt,
        tokenize=True,
        add_generation_prompt=True,
        return_tensors="pt",
        padding=True
    )
    return enc
```

```
def predict_label(model, tokenizer, text, max_new_tokens=5):
    prompt = make_prompt(text)
    enc = tokenize_prompt(tokenizer, prompt).to(model.device)

    with torch.no_grad():
        output = model.generate(
            input_ids=enc,
            max_new_tokens=max_new_tokens,
            do_sample=False
        )

    prompt_len = enc.shape[-1]

    decoded = tokenizer.decode(
        output[0, prompt_len:],
        skip_special_tokens=True
    ).strip().lower()

    for label in labels.keys():
        if label in decoded:
            return label

    return "unknown"
```

```
def evaluate(df, model, tokenizer, n_samples=100):
    correct = 0
    total = 0

    for _, row in df.sample(n_samples).iterrows():
        pred = predict_label(model, tokenizer, row["text"], max_new
        gold = row["label_text"]

        if pred == gold:
            correct += 1
        total += 1

    return correct / total
```

```
acc_before = evaluate(df, model, tokenizer, n_samples=1)
print("Accuracy before fine-tuning:", acc_before)
```

its taking way too looong, so skipping it for now

```
ip = tokenizer(['hello how are'], return_tensors = 'pt').to(device)
ip
```

```
{'input_ids': tensor([[    1, 22172,   920,   526]],
device='cuda:0'), 'attention_mask': tensor([[1, 1, 1, 1]],
device='cuda:0')}
```

```
res = model(input_ids = ip['input_ids'])
res
```

```
CausalLMOutputWithPast(loss=None, logits=tensor([[[ -4.8438,
0.7969,   4.3750,  ...,  -5.2188,  -2.4531,  -4.2188],
         [-10.1250, -10.1875,   6.0312,  ...,  -4.2188,  -8.6875,
-2.4531],
         [ -8.1875,  -8.3750,   8.4375,  ...,  -3.4844,  -6.7500,
-2.5781],
         [ -7.0625,  -6.7812,   7.9688,  ...,  -2.8906,  -6.5938,
-3.1094]]],
       device='cuda:0', dtype=torch.bfloat16, grad_fn=
<UnsafeViewBackward0>), past_key_values=DynamicCache(layers=
[DynamicLayer, DynamicLayer, DynamicLayer, DynamicLayer,
DynamicLayer, DynamicLayer, DynamicLayer, DynamicLayer,
DynamicLayer, DynamicLayer, DynamicLayer, DynamicLayer,
DynamicLayer, DynamicLayer, DynamicLayer, DynamicLayer,
DynamicLayer, DynamicLayer, DynamicLayer, DynamicLayer,
DynamicLayer, DynamicLayer]), hidden_states=None, attentions=None)
```

```
# WOOOHHH , THESE ARE LOGITS!! LETS SEE WHAT IS IT EXACTLY
```

```
ip['input_ids']
```

```
tensor([[    1, 22172,   920,   526]], device='cuda:0')
```

```
res.logits.shape # torch.Size([1, 4, 32000]) means logits for ip['i
```

```
torch.Size([1, 4, 32000])
```

```
import torch.nn as nn
```

```
for tens in res.logits[0]:
  prob_dist = nn.Softmax()(tens)
  pred_tok = prob_dist.argmax()
  print(tokenizer.decode(pred_tok))
```

```
<
,
are
you
/usr/local/lib/python3.12/dist-packages/torch/nn/modules/module.py:1
  return self._call_impl(*args, **kwargs)
```

⚠️ Only the LAST one is the model's actual answer. The earlier ones are internal next-token predictions, not outputs.

```
model
```

```
LlamaForCausalLM(
  (model): LlamaModel(
    (embed_tokens): Embedding(32000, 2048)
    (layers): ModuleList(
      (0-21): 22 x LlamaDecoderLayer(
        (self_attn): LlamaAttention(
          (q_proj): Linear(in_features=2048, out_features=2048,
bias=False)
          (k_proj): Linear(in_features=2048, out_features=256,
bias=False)
          (v_proj): Linear(in_features=2048, out_features=256,
bias=False)
          (o_proj): Linear(in_features=2048, out_features=2048,
bias=False)
        )
        (mlp): LlamaMLP(
          (gate_proj): Linear(in_features=2048, out_features=5632,
bias=False)
          (up_proj): Linear(in_features=2048, out_features=5632,
bias=False)
          (down_proj): Linear(in_features=5632, out_features=2048,
bias=False)
          (act_fn): SiLUActivation()
        )
        (input_layernorm): LlamaRMSNorm((2048,), eps=1e-05)
        (post_attention_layernorm): LlamaRMSNorm((2048,), eps=1e-
05)
      )
    )
    (norm): LlamaRMSNorm((2048,), eps=1e-05)
    (rotary_emb): LlamaRotaryEmbedding()
  )
  (lm_head): Linear(in_features=2048, out_features=32000,
bias=False)
)
```

lets us now see what would have been given by .generate

```
dot_generate_op = model.generate(ip['input_ids'], max_new_tokens=1)
dot_generate_op
```

```
tensor([[    1, 22172,   920,   526,   366]])
```

```
tokenizer.batch_decode(dot_generate_op)
```

```
['<s> hello how are you']
```

This is AWESOMEEE

```
op = model.generate(input_ids=tokenizer(['with great power comes gr
```

```
tokenizer.batch_decode(op)
```

```
['<s> with great power comes great responsibility']
```

lets try fine tuning -> model to predict responsibili-TEA instead

```
prompts = [
    {'role': 'system', 'content': 'you are a 200 IQ person'},
    {'role': 'user', 'content': 'With great power comes great'},
    {'role': 'assistant', 'content': ''}, # we can do this but then
]

chat_template = tokenizer.apply_chat_template(prompts, add_generati
chat_template
```

```
'<|system|>\nyou are a 200 IQ person</s>\n<|user|>\nWith great powe
r comes great</s>\n<|assistant|>\n</s>\n'
```

```
ans = ' responsibili-TEA'
```

```
full_res_text = chat_template + " " + ans + tokenizer.eos_token
full_res_text
```

```
'<|system|>\nyou are a 200 IQ person</s>\n<|user|>\nWith great power comes great</s>\n<|assistant|>\n</s>\n  responsibili-TEA</s>'
```

```
full_res_ip_ids = tokenizer(full_res_text, return_tensors = 'pt')['
full_res_ip_ids
```

```
tensor([[    1,   529, 29989,  5205, 29989, 29958,    13,  6293,
526,   263,
         29871, 29906, 29900, 29900,   306, 29984,  2022,     2,
29871,    13,
         29966, 29989,  1792, 29989, 29958,    13,  3047,  2107,
3081,  5304,
          2107,     2, 29871,    13, 29966, 29989,   465, 22137,
29989, 29958,
            13,     2, 29871,    13, 29871,  5544,   747,  2638,
29899,  4330,
         29909,     2]])
```

```
input_ids = full_res_ip_ids
input_prompt_ids = input_ids[:, :-1] # start to end-1
output_prompt_ids = input_ids[:, 1:] # start+1 to end
```

but we'll be applying loss function on just the tokens which are part of the "answer" i.e generated tokens so lets extract the "answer" from the full response

```
chat_temp_ip_ids = tokenizer(chat_template, return_tensors='pt')['i
chat_temp_ip_ids # these are tokens without answers
```

```
tensor([[    1,   529, 29989,  5205, 29989, 29958,    13,  6293,
526,   263,
         29871, 29906, 29900, 29900,   306, 29984,  2022,     2,
29871,    13,
         29966, 29989,  1792, 29989, 29958,    13,  3047,  2107,
3081,  5304,
          2107,     2, 29871,    13, 29966, 29989,   465, 22137,
29989, 29958,
            13,     2, 29871,    13]])
```

```
# we need to extract output tokens which are here:  5544,   747,  2
```

```
expected_op_ip_ids = full_res_ip_ids[0][chat_temp_ip_ids[0].shape[0
expected_op_ip_ids
```

```
tensor([29871,  5544,   747,  2638, 29899,  4330, 29909,     2])
```

```
tokenizer.decode(expected_op_ip_ids)
```

```
' responsibili-TEA</s>'
```

```
padded_expected_op_ids = tokenizer([' responsibili-TEA</s>'], paddi
padded_expected_op_ids
```

```
tensor([[    2,     2,     2,     2,     2,     2,     2,     2,
2,     2,
             2,     2,     2,     2,     2,     2,     2,     2,
2,     2,
             2,     2,     2,     2,     2,     2,     2,     2,
2,     2,
             2,     2,     2,     2,     2,     2,     2,     2,
2,     2,
             2,     2,     2, 29871,  5544,   747,  2638, 29899,
4330, 29909,
             2]])
```

```
padded_expected_op_ids.shape, output_prompt_ids.shape
```

```
(torch.Size([1, 51]), torch.Size([1, 51]))
```

```
# but we are working with tensors -> rectuangular shapes -> so it s
```

```
output_prompt_ids.shape, expected_op_ip_ids.shape
```

```
(torch.Size([1, 51]), torch.Size([8]))
```

```
tokenizer.convert_ids_to_tokens(2), tokenizer.convert_ids_to_tokens
```

```
('</s>', '<s>')
```

```
masked_padded_expected_tokens = torch.where(padded_expected_op_ids
```

```
masked_padded_expected_tokens # why -100? because trainer ignores t
```

```
tensor([[ -100,  -100,  -100,  -100,  -100,  -100,  -100,  -100,
-100,  -100,
          -100,  -100,  -100,  -100,  -100,  -100,  -100,  -100,
-100,  -100,
          -100,  -100,  -100,  -100,  -100,  -100,  -100,  -100,
-100,  -100,
          -100,  -100,  -100,  -100,  -100,  -100,  -100,  -100,
-100,  -100,
          -100,  -100,  -100, 29871,  5544,   747,  2638, 29899,
4330, 29909,
          -100]])
```

```
# lets stich this into one function
```

```
import torch

def preprocess_data(input_prompts, target_responses, tokenizer):
    # input_prompts is assumed to be a list of conversations (each
    # apply_chat_template can handle a batch of conversations
    chat_templates = tokenizer.apply_chat_template(input_prompts, c

    # Create full response texts by appending target_responses
    # list of strings
    full_response_texts = [
        chat_template + " " + target_response + tokenizer.eos_token
        for chat_template, target_response in zip(chat_templates, t
    ]

    # Tokenize full response texts and create attention mask
    # Using padding=True to handle varying lengths in a batch
    tokenized_full_responses = tokenizer(full_response_texts, retur
    input_ids = tokenized_full_responses['input_ids']
    attention_mask = tokenized_full_responses['attention_mask']

    # Create labels for language modeling, initially a clone of inp
    labels = input_ids.clone()

    # Mask out prompt tokens in labels (set them to -100)
    # These tokens should not contribute to the loss
    for i, chat_template_str in enumerate(chat_templates):
        # Tokenize the chat_template_str to get its length without
        # This provides the exact length of the prompt part
        prompt_token_ids = tokenizer(chat_template_str, return_tens
        prompt_len = prompt_token_ids.shape[1]

        # Set tokens corresponding to the prompt in `labels` to -10
        labels[i, :prompt_len] = -100

    # Ensure padding tokens in labels are also -100
    # Padding tokens should not contribute to the loss either
    labels[attention_mask == 0] = -100

    return {
        'input_ids': input_ids,
        'labels': labels, # Renamed to 'labels' for HuggingFace Tra
        'attention_mask': attention_mask
    }
```

```
input_prompts = [[{'role': 'system', 'content': 'you are a 200 IQ p
 {'role': 'user', 'content': 'With great power comes great'},
 {'role': 'assistant', 'content': ''}]]
target_response = [' responsibili-TEA']
data = preprocess_data(input_prompts, target_response, tokenizer)
data
```

```
{'input_ids': tensor([[  529, 29989,  5205, 29989, 29958,    13,
6293,   526,   263, 29871,
         29906, 29900, 29900,   306, 29984,  2022,     2, 29871,
13, 29966,
         29989,  1792, 29989, 29958,    13,  3047,  2107,  3081,
5304,  2107,
             2, 29871,    13, 29966, 29989,   465, 22137, 29989,
29958,    13,
         29871,  5544,   747,  2638, 29899,  4330, 29909,
2]]),
 'labels': tensor([[ -100,  -100,  -100,  -100,  -100,  -100,
-100,  -100,  -100,  -100,
          -100,  -100,  -100,  -100,  -100,  -100,  -100,  -100,
-100,  -100,
          -100,  -100,  -100,  -100,  -100,  -100,  -100,  -100,
-100,  -100,
          -100,  -100,  -100,  -100,  -100,  -100,  -100,  -100,
-100,  -100,
         29871,  5544,   747,  2638, 29899,  4330, 29909,
2]]),
 'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1]])}
```

```
data = {k: v.to(device) for k, v in data.items()}
```

Calc Loss

```
out = model(input_ids = data['input_ids'])
out.logits.shape
```

```
torch.Size([1, 48, 32000])
```

```
# just checking something
temp = ''
for tens in out.logits[0]:
  prob_dist = nn.Softmax()(tens)
  pred_tok = prob_dist.argmax()
  temp += tokenizer.decode(pred_tok)
print(temp)
```

```
system|>
</s>areagreat
010Qgenius.
<|user|>
Canaintelligencecomesgreatresponsibility
<|assistant|>
YesWithibilitiesy
E-
```

```
data['labels'], data['labels'].shape
```

```
(tensor([[ -100,  -100,  -100,  -100,  -100,  -100,  -100,  -100,
-100,  -100,
           -100,  -100,  -100,  -100,  -100,  -100,  -100,  -100,
-100,  -100,
           -100,  -100,  -100,  -100,  -100,  -100,  -100,  -100,
-100,  -100,
           -100,  -100,  -100,  -100,  -100,  -100,  -100,  -100,
-100,  -100,
          29871,  5544,   747,  2638, 29899,  4330, 29909,     2]],
        device='cuda:0'),
 torch.Size([1, 48]))
```

```
import torch.nn as nn
def cal_loss(logits, labels):
  loss_fn = nn.CrossEntropyLoss(reduction="none")
  loss = loss_fn(logits.view(-1, logits.shape[-1]), labels.view(-1)
  return loss
```

when reduction="none" So conceptually, the raw result is always something like: [loss_0, loss_1, ..., loss_47]

otherwise it aggregates losses into single scalar default is reduction="mean"

```
cal_loss(out.logits, data['labels']) # -> returns a tensor of loss
```

```
tensor([ 0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
0.0000,  0.0000,
         0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
0.0000,  0.0000,
         0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
0.0000,  0.0000,
         0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
0.0000,  0.0000,
         0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
0.0000,  0.0000,
        10.3125, 16.5000, 14.0000, 11.4375,  9.0000, 10.0625,
12.1875, 19.7500],
       device='cuda:0', dtype=torch.bfloat16, grad_fn=
<NllLossBackward0>)
```

Time to FineTune

```
from torch.optim import AdamW
```

```
data
```

```
{'input_ids': tensor([[  529, 29989,  5205, 29989, 29958,    13,
6293,   526,   263, 29871,
          29906, 29900, 29900,   306, 29984,  2022,     2, 29871,
13, 29966,
          29989,  1792, 29989, 29958,    13,  3047,  2107,  3081,
5304,  2107,
              2, 29871,    13, 29966, 29989,   465, 22137, 29989,
29958,    13,
          29871,  5544,   747,  2638, 29899,  4330, 29909,     2]],
        device='cuda:0'),
 'labels': tensor([[ -100,  -100,  -100,  -100,  -100,  -100,
-100,  -100,  -100,  -100,
           -100,  -100,  -100,  -100,  -100,  -100,  -100,  -100,
-100,  -100,
           -100,  -100,  -100,  -100,  -100,  -100,  -100,  -100,
-100,  -100,
           -100,  -100,  -100,  -100,  -100,  -100,  -100,  -100,
-100,  -100,
          29871,  5544,   747,  2638, 29899,  4330, 29909,     2]],
        device='cuda:0'),
 'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1]],
        device='cuda:0')}
```

```
optimizer = AdamW(model.parameters(), lr=1e-4, weight_decay=0.01)
```

```
model.parameters()
```

```
<generator object Module.parameters at 0x799d3302b840>
```

```
for _ in range(10):
  optimizer.zero_grad()

  out = model(input_ids=data['input_ids'])
  loss = cal_loss(out.logits, data['labels']).mean()

  loss.backward()
  optimizer.step()

  print("loss = ", loss.item())
  del out, loss
```

```
loss =  2.15625
loss =  0.50390625
loss =  0.1328125
loss =  0.208984375
loss =  0.00311279296875
loss =  0.000644683837890625
loss =  0.0002536773681640625
loss =  0.0002574920654296875
loss =  0.0003910064697265625
loss =  0.0002651214599609375
```

```
# WOOAHHH, LETS SEE WHAT 'IT' HAS LEARNT!!
```

```
tokenizer.batch_decode(model.generate(data['input_ids'], max_new_tc
```

```
['<|system|>\nyou are a 200 IQ person</s> \n<|user|>\nWith great
power comes great</s> \n<|assistant|>\n  responsibili-TEA</s></s>']
```

```
# yayyyyyyyyyyy
```

Now lets do this using LoRA

```
# running this to clear GPU ram
del model, optimizer, data
torch.cuda.empty_cache()
```

```
# refreshing model weights
```

```
import torch
model = AutoModelForCausalLM.from_pretrained(model_id, torch_dtype
```

```
data = {k: v.to(device) for k, v in data.items()}
```

```
from peft import LoraConfig, get_peft_model
```

```
config = LoraConfig(
    task_type = 'CAUSAL_LM',
    r = 8,
    lora_alpha = 32,
    lora_dropout = 0.1,
    target_modules='q_proj v_proj'.split()
)
```

```
lora_model = get_peft_model(model, config)
```

```
lora_model.print_trainable_parameters()
```

```
trainable params: 1,126,400 || all params: 1,101,174,784 || trainabl
```

```
for _ in range(10):
  optimizer.zero_grad()

  out = lora_model(input_ids=data['input_ids'])
  loss = cal_loss(out.logits, data['labels']).mean()

  loss.backward()
  optimizer.step()

  print("loss = ", loss.item())
  del out, loss
```

```
loss =  2.15625
loss =  2.078125
loss =  1.984375
loss =  1.8671875
loss =  1.7578125
loss =  1.6484375
loss =  1.5625
loss =  1.4609375
loss =  1.390625
loss =  1.3125
```

```
tokenizer.batch_decode(lora_model.generate(data['input_ids']))
```

```
['<|system|>\nyou are a 200 IQ person</s> \n<|user|>\nWith great
power comes great</s> \n<|assistant|>\n  responsibili-TEA</s><s>
<|system|>\n</s>']
```

```
# double baaammmmm!
```

```
# btw -> GPU usage was 11 gigs while finetuning full model
# and just 6 gigs while finetuning LoRA model
```