In [ ]:

```
"This is the code used to tune the hyperparameters."
```

In [1]:

```
#Libaries
from sklearn.model_selection import train_test_split
import pandas as pb
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, TrainingArguments, Trainer
, pipeline
from ray import tune
from ray.tune.search.hyperopt import HyperOptSearch
from ray.tune.schedulers import ASHAScheduler
import math

#Modell och tokenizer
model_name = "AI-Sweden-Models/gpt-sw3-126m"
task_name = "testNamn"
tokenizer = AutoTokenizer.from_pretrained(model_name,padding_side = 'left')
def model_init():
    return AutoModelForCausalLM.from_pretrained(
        model_name, return_dict=True,)
device = "cuda:0" if torch.cuda.is_available() else "cpu"
model_init().to(device)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[1], line 6
      4 import torch
      5 from transformers import AutoTokenizer, AutoModelForCausalLM, TrainingArguments,
Trainer, pipeline
----> 6 from ray import tune
      7 from ray.tune.search.hyperopt import HyperOptSearch
      8 from ray.tune.schedulers import ASHAScheduler

ModuleNotFoundError: No module named 'ray'
```

In [ ]:

```
#Takes away all columns that have less than one procent of its boxes filled.
def taBortEnProcent(FromFile, PlaceToSave):
    data = pb.read_excel(FromFile)
    missing_percentage = (data.isnull().sum() / len(data)) * 100
    cols_to_drop = missing_percentage[missing_percentage >= 99].index
    datasmall_filtered = data.drop(columns=cols_to_drop)

    print("Shape after filtering:", datasmall_filtered.shape)

    datasmall_filtered.to_excel(PlaceToSave, index=False)

    return datasmall_filtered
```

In [ ]:

```
#This function takes away several keywords and it's sentence in the output texts.

def rensaUtdata(FromFile):
    data = FromFile

    print(data.shape)

    input_tokens = data.iloc[:, 1:].values.tolist()
    output_tokens = data.iloc[:, 0].values.tolist()
    output_tokens = [str(value) for value in output_tokens]

    processed_texts = []
```

```
    for text in output_tokens:
        SokOrd = {'jmf','Jmf','JMF','jämfört','Jämfört','jämförelse','Jämförelse','tidig
are','Tidigare','föregående','Föregående','2010','2011','2012','2013','2014','2015','201
6','2017','2018','2019','2020','2021','2022','2023','2024'}

        for i in range(0, 3):
            for s in SokOrd:
                head, sep, tail = text.partition(s)
                if head == text:
                    head = ""
                txt = head[::-1]

                for tecken in txt:
                    if tecken == '.':
                        head2, sep2, tail2 = tail.partition('.')
                        Second_head, Second_sep, Second_tail = txt.partition(tecken)
                        new_txt = Second_tail[::-1]
                        text = new_txt + tail2
                        break

                    if tecken == '(':
                        head2, sep2, tail2 = tail.partition(')')
                        Second_head, Second_sep, Second_tail = txt.partition(tecken)
                        Second_tail = Second_tail[1:] #takes away the last whitespace
                        new_txt = Second_tail[::-1]
                        text = new_txt + tail2
                        break

                    if tecken == '?':
                        head2, sep2, tail2 = tail.partition('.')
                        Second_head, Second_sep, Second_tail = txt.partition(tecken)
                        Second_tail = Second_tail[1:] #takes away the last whitespace
                        new_txt = Second_tail[::-1]
                        text = new_txt + tail2
                        break



        processed_texts.append(text.strip())

    output_tokens = processed_texts


    return {
        "output": output_tokens,
        "input": input_tokens,
    }
```

In [ ]:

```
#Order all input and output text in to pairs. Then formating them in order for the model
to understand.
def formatering(indata, utdata):
    par_tokens = [(utdata[i], indata[i])for i in range(len(indata))]

    train_texts, val_and_test_texts = train_test_split(par_tokens, test_size=0.2)
    val_texts, test_texts = train_test_split(val_and_test_texts, test_size=0.5)

    output_from_par_train_small = [str(item[0]) for item in train_texts]
    output_from_par_val_small = [str(item[0]) for item in val_texts]
    input_from_par_train_small = [str(item[1]) for item in train_texts]
    input_from_par_val_small = [str(item[1]) for item in val_texts]

    formatted_data_train = [f"<|endoftext|><s>User: Skriv en patientjornal efter en ultra
ljudsundersökning utifrån dessa värden: {input_token}<s>Bot:{output_token}<s>"
                        for input_token, output_token in zip(input_from_par_train_small,
output_from_par_train_small)]

    formatted_data_val = [f"<|endoftext|><s>User: Skriv en patientjornal efter en ultralj
udsundersökning utifrån dessa värden: {input_token}<s>Bot:{output_token}<s>"
```

```
                        for input_token, output_token in zip(input_from_par_val_small, o
utput_from_par_val_small)]


    return {
        "train": formatted_data_train,
        "val": formatted_data_val,
    }
```

In [ ]:

```
#Creates a dataset and tokenize the texts.

class MyDataset(torch.utils.data.Dataset):
    def __init__(self, formatted_data, tokenizer):
        self.formatted_data = formatted_data
        self.tokenizer = tokenizer

    def __len__(self):
        return len(self.formatted_data)

    def __getitem__(self, idx):
        formatted_data = self.formatted_data[idx]

        inputs = self.tokenizer.encode_plus(
            formatted_data,
            return_tensors='pt',
            padding='max_length',
            truncation=True,
            max_length = 2048,
            return_attention_mask=True,
        )

        return {
            "input_ids": inputs.input_ids.flatten(),
            "labels": inputs.input_ids.flatten(),
            "attention_mask": inputs.attention_mask.flatten(),
        }
```

In [ ]:

```
#starts hyparparameter tuning with ASHAS
def StartHyperParamTuning(train_data, val_data):

    for i in range(0, 3):
        for j in range(0, 2):
            for s in range(0,3):

                if j == 0: activationfunction = "gelu"
                if j == 1: activationfunction = "swish"

                def model_init():
                    return AutoModelForCausalLM.from_pretrained(
                        model_name,
                        return_dict=True,
                        n_layer = int(6 * math.pow(2,i)),
                        n_head = int(6 * math.pow(2,s)),
                        activation_function = activationfunction,

                    )

                device = "cuda:0" if torch.cuda.is_available() else "cpu"
                model_init().to(device)


                training_args = TrainingArguments(
                "test", evaluation_strategy="steps",
                eval_steps=10,
                do_train=True,
                do_eval=True,
                per_device_eval_batch_size=4,
```

```
                per_device_train_batch_size=3,
                disable_tqdm=True
                )
                trainer = Trainer(
                model = None,
                args=training_args,
                tokenizer=tokenizer,
                train_dataset=train_data,
                eval_dataset=val_data,
                model_init=model_init,
                )

                def ray_hp_space(*args, **kwargs):
                    return {
                        "per_device_train_batch_size": tune.choice([1,2]),
                        "learning_rate": tune.loguniform(1e-6, 1e-4),
                        "gradient_accumulation_steps": tune.choice([1,2,3,4]),
                        "num_train_epochs": tune.choice([1,2,3]),
                        "gradient_checkpointing" : tune.choice([False]),
                        "weight_decay": tune.uniform(0.0, 0.2),
                        "fp16": tune.choice([True, False]),
                        "warmup_ratio" : tune.uniform(0, 1e-6),
                        "adam_beta1": tune.uniform(0.4, 0.999999999),
                        "adam_beta2": tune.uniform(0.4, 0.999999999),
                        "max_grad_norm": tune.uniform(0.2, 1.5),
                        "fp16_opt_level": tune.choice(["O1", "O2"]),
                        "adam_epsilon": tune.loguniform(1e-10, 1e-5),

                    }


                bestParamNow =[{
                    "per_device_train_batch_size": 1,
                    "learning_rate": 7.65391e-05,
                    "gradient_accumulation_steps": 1,
                    "num_train_epochs": 2,
                    "gradient_checkpointing" : False,
                    "weight_decay": 0.0704687,
                    "fp16": True,
                    "warmup_ratio": 4.54937e-07,
                    "adam_beta1": 0.9,
                    "adam_beta2": 0.999,
                    "max_grad_norm": 1,
                    "fp16_opt_level": "O1",
                    "adam_epsilon": 1e-8,
                }]

                best_trial = trainer.hyperparameter_search(
                    direction="minimize",
                    hp_space = ray_hp_space,
                    backend="ray",
                    n_trials=64,
                    search_alg=HyperOptSearch(metric="eval_loss", mode="min", points_to_
evaluate = bestParamNow),
                    scheduler=ASHAScheduler(metric="eval_loss", mode="min"))

                print("Antal lager: ",  6 * math.pow(2,i))
                print("Antal heads: ",  6 * math.pow(2,s))
                print("Activationfunction: ", activationfunction)
                print("Best Validation Loss: ", best_trial[1])
                print("Corresponding Hyperparameters: ", best_trial[2])

                with open("HyperOpti.txt", "a") as file:
                    file.write(("Antal lager: " + str(6 * math.pow(2,i))) +"\n")
                    file.write(("Antal heads: " + str(6 * math.pow(2,s))) +"\n")
                    file.write("Activationfunction: " +str(activationfunction) + "\n")
                    file.write("Best Validation Loss: " + str(best_trial[1]) + "\n")
                    file.write("Corresponding Hyperparameters: " + str(best_trial[2]) +
"\n")
```

In [ ]:

```
#Process för att träna en modell
'''
As you can see in the following function a new file is saved after columns with only 1 %
filled have been taken away.
Only needs to be ran once, saves new file to later use
Use a small dataset when hypertuning, otherwise it takes a very long time
'''
file = taBortEnProcent("place_of_original_file", "place_to_save_file")
utdata = rensaUtdata(file)["output"]
indata = rensaUtdata(file)["input"]
train_texter = formatering(indata, utdata)["train"]
val_texter = formatering(indata, utdata)["val"]

train_dataset = MyDataset(train_texter, tokenizer)
val_dataset = MyDataset(train_texter, tokenizer)

StartHyperParamTuning(train_dataset, val_dataset)
```