In [ ]:

```
#Libaries
from sklearn.model_selection import train_test_split
import pandas as pb
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, TrainingArguments, Trainer
, pipeline

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

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
    data_filtered = data.drop(columns=cols_to_drop)

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

    data_filtered.to_excel(PlaceToSave, index=False)

    return data_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]
    output_from_par_test_small = [str(item[0]) for item in test_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]
    input_from_par_test_small = [str(item[1]) for item in test_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)]

    formatted_data_test = [f"<|endoftext|><s>User: Skriv en patientjornal efter en ultral
judsundersökning utifrån dessa värden: {input_token}<s>Bot:{output_token}<s>"
                        for input_token, output_token in zip(input_from_par_test_small,
output_from_par_test_small)]


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

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 a training-loop and after saves the model and tokenizer to a desierd file.

def StartaTrain(model, tokenizer, train_dataset, val_dataset, PlaceToSave):

    training_args = TrainingArguments(
        output_dir="test_trainer",
        evaluation_strategy="steps",
        eval_steps=31100,
        per_device_train_batch_size= 1,
        learning_rate= 7.65391e-05,
        gradient_accumulation_steps = 1,
        num_train_epochs= 3,
        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= 1.0,
        fp16_opt_level= 'O1',
        adam_epsilon= 1e-08,
        logging_steps=31100,
        save_strategy= "steps",
        save_steps=31100,
        logging_dir=("./logs"),
    )

    trainer = Trainer(
        model = model,
        tokenizer= tokenizer,
        args = training_args,
        train_dataset= train_dataset,
        eval_dataset= val_dataset,
    )

    trainer.train()
    trainer.save_model(PlaceToSave)
    tokenizer.save_pretrained(PlaceToSave)
```

In [ ]:

```
#Process för att träna en modell
'''
As you can see in the below functione a new file is saved after 1 % have been taken away.
So if you alredy have doen this function one time, its more time efficent
```

```
to just loade the new file directly
'''
file = taBortEnProcent("From_this_file", "To_this_file")
utdata = rensaUtdata(file)["output"]
indata = rensaUtdata(file)["input"]
train_texter = formatering(indata, utdata)["train"]
val_texter = formatering(indata, utdata)["val"]
test_texter = formatering(indata, utdata)["test"]

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

StartaTrain(model, tokenizer, train_dataset, val_dataset, "Place_to_save")
```

In [ ]:

```
'''
The folowing code is to generate an answer from the AI. We have created a seperate file f
or this but the code
is a litle bit diffrent when you use the same dataset as when you train.

So this code should be used for investigation purposes when you dont want or have
the time to load a new dataset. The other file, "Kex_AI_Generating" is designed to be use
d
when you want too create several texts.
'''
```

In [ ]:

```
def generating(text, model, tokenizer, device):
    index_bot = text.find("Bot")
    new_string = text[:index_bot + 4]
    prompt = new_string.strip()


    token_count = len(tokenizer.encode_plus(prompt)["input_ids"])
    max_token_count = 2048 - 140

    if token_count > max_token_count:
        # Hitta indexet där de tre sista elementen börjar
        end_index = len(prompt) - 7
        end_seq = prompt[end_index:]
        tokens = tokenizer.encode_plus(prompt[:end_index])["input_ids"]
        tokens = tokens[:max_token_count]
        prompt = (tokenizer.decode(tokens) + (end_seq)).strip()


    generator = pipeline('text-generation', tokenizer=tokenizer, model=model, device=dev
ice)
    generated = generator(prompt, max_new_tokens=140, do_sample=True, temperature=0.47,
top_p=1, top_k = 23, repetition_penalty = 1.05)[0]["generated_text"]
    return generated
```

In [ ]:

```
#process to generate text
device = "cuda:0" if torch.cuda.is_available() else "cpu"
tokenizer = AutoTokenizer.from_pretrained("/mnt/d1/KEX/.myenv/GPT-sw3-356m-BaseTest_myown
")
model = AutoModelForCausalLM.from_pretrained("/mnt/d1/KEX/.myenv/GPT-sw3-356m-BaseTest_my
own")
model.eval()
model.to(device)

file = pb.read_excel("LitenData_rensad.xlsx")
in_data = rensaUtdata(file)["input"]
out_data = rensaUtdata(file)["output"]
test_dataset = formatering(in_data, out_data)["test"]
idx = 3
generated_text = generating(test_dataset[idx], model, tokenizer, device)
print(generated_text)
```