KIDS SCIENCE BOT

GRADUATION PROJECT

CONTENT

01 PROBLEM DEFINITION

02 INTRODUCTION

03 STRATEGY

04 EXECUTION

05 ANALYSIS & OBSERVATION

06 OUTRO

PROBLEM DEFINITION

We wanted to create a bot that can help kids find answers for science related questions, which can help boost their knowledge and exploration of the universe.

GOALS AND OBJECTIVES

Objective n° 1

A young kid can ask the chatbot a science related question the bot will generate a response with the answer.

Objective n° 2

The Bot can identify the type of question as science or non science and only respond in case of a science question and decline politely if it’s not.

Objective n° 3

The Bot generate an answer that is concise and straight to the point.

STRATEGY

In order to achieve our objectives we need to ensure that the bot can identify scientific questions and only answer those

We use a transformer and traine it on “Trec “ Dataset after labeling it , as science and non science questions and train the model on a classification task.

STEP 1

We integrate the classification trained model with a LLM for answer generation.

STEP 2

We integrate a UI for ease of use and dockerize the project for deployment

STEP 3

EXECUTION PARAMETERS

BERT-BASE-UNCASED

TRANSFOMER

Flan-T5 large

GEN. MODEL

Stremlit

USER INTERFACE

EXECUTION

STEP 1 - DATA PROCESSING

- Loading Trec DataSet
- Labeling the Data and then map the labels

We applied the mapping to train, validation and test sets

```python
from datasets import load_dataset

# Loading the trec dataset
dataset = load_dataset('trec')
```

```python
# labeling data , targeting sciene and non science

science_fine_labels = [
    'ENTY:animal', 'ENTY:body', 'ENTY:disease', 'ENTY:plant',
    'ENTY:substance', 'ENTY:technique', 'NUM:period', 'NUM:weight',
    'NUM:speed', 'NUM:temperature', 'NUM:distance', 'NUM:volume',
    'NUM:size', 'LOC:planet', 'DESC:definition', 'DESC:reason'
    # we can add more labels as we need
]

def map_label(example):
    fine_label = dataset['train'].features['fine_label'].names[example['fine_label']]
    if fine_label in science_fine_labels:
        example['label'] = 1  # Science
    else:
        example['label'] = 0  # Non-science
    return example
```

```python
# we will apply the mapping to the train, validation, and test sets
dataset = dataset.map(map_label)
```

```
Map: 100%  5452/5452 [00:01<00:00, 4451.29 examples/s]
Map: 100%  500/500 [00:00<00:00, 4231.02 examples/s]
```

EXECUTION

STEP 2 - MODEL PREPARATION

- Created Tokenization function
- Created Attention Masks
- Loaded Sequence Classification Model

- We moved the model to GPU for training performance

```python
# loading bert base uncased transfomer to use it for training
from transformers import BertTokenizerFast

tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
```

```python
# creating a tokenization function for the data

def tokenize_function(example):
    return tokenizer(
        example['text'],
        truncation=True,
        padding='max_length',
        max_length=128
    )

train_dataset = train_dataset.map(tokenize_function, batched=True)
valid_dataset = valid_dataset.map(tokenize_function, batched=True)
test_dataset = test_dataset.map(tokenize_function, batched=True)
```

```python
# creating attention masks for train , validation and test

train_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'label'])
valid_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'label'])
test_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'label'])
```

```python
# loading the sequence classification model

from transformers import BertForSequenceClassification

model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)
```

```python
# moving model to GPU
import torch

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
```

EXECUTION

STEP 3 - MODEL TRAINING

- Batch Size : 16
- Epochs : 5
- Learning Rate : 2e-5
- Metric : Accuracy

- We loaded the Best model at end with Accuracy : 97.6%
- Training Loss : 2.7%
- Validation Loss : 1.4%

```python
# training the classification model on the data

from transformers import TrainingArguments
import numpy as np
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from transformers import Trainer

training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    evaluation_strategy='epoch',
    save_strategy='epoch',
    logging_dir='./logs',
    learning_rate=2e-5,
    load_best_model_at_end=True,
    metric_for_best_model='accuracy',
    greater_is_better=True,
)

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=1)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, predictions, average='binary')
    acc = accuracy_score(labels, predictions)
    return {
        'accuracy': acc,
        'f1': f1,
        'precision': precision,
        'recall': recall
    }



trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
)
trainer.train()
trainer.evaluate(test_dataset)
trainer.save_model('/content/drive/MyDrive/Depi Project/kids_science_bot/science_classifier_model')
tokenizer.save_pretrained('/content/drive/MyDrive/Depi Project/kids_science_bot/science_classifier_model')
```

```
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1525: FutureWarning: `evaluation_strategy
  warnings.warn(
[1535/1535 09:00, Epoch 5/5]
```

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	No log	0.172017	0.952381	0.754717	0.800000	0.714286
2	0.150000	0.138584	0.974359	0.862745	0.956522	0.785714
3	0.150000	0.160215	0.976190	0.873786	0.957447	0.803571
4	0.027200	0.144163	0.976190	0.876190	0.938776	0.821429
5	0.006300	0.170689	0.976190	0.873786	0.957447	0.803571

```
[35/35 00:03]
```

EXECUTION

STEP 4 - LOADING MODEL

- Loaded our Classifier model
- Loaded Flan-T5-Large Generative Model

- We created a function to Classify the question.

```python
# we load the classifier model
classifier_model_path = '/content/drive/MyDrive/Depi Project/kids_science_bot/science_classifier_model'
classifier_tokenizer = BertTokenizerFast.from_pretrained(classifier_model_path)
classifier_model = BertForSequenceClassification.from_pretrained(classifier_model_path)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
classifier_model.to(device)
classifier_model.eval()

# we will load a generative model for answer generation
generator_tokenizer = T5Tokenizer.from_pretrained('google/flan-t5-large')
generator_model = T5ForConditionalGeneration.from_pretrained('google/flan-t5-large')
generator_model.to(device)
generator_model.eval()

# we create a function to classify the question

def is_science_question(question):
    inputs = classifier_tokenizer(
        question,
        return_tensors='pt',
        truncation=True,
        padding=True,
        max_length=128
    ).to(device)

    with torch.no_grad():
        outputs = classifier_model(**inputs)
    logits = outputs.logits
    probabilities = torch.softmax(logits, dim=1)
    predicted_label = torch.argmax(probabilities, dim=1).item()

    # label 1 is for science and 0 for non science
    return predicted_label == 1
```

EXECUTION

STEP 4 - LOADING MODEL

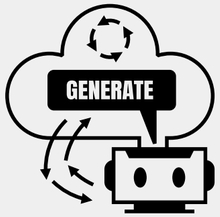
- We added another keyword classification layer to ensure flexibility of the model

- we included a wide array of science-related keywords and defined a function to handle them

```python
# we will list science keywords to use them as an extra layer of keyword based classification
science_keywords = [
    "gravity", "atom", "molecule", "photosynthesis", "evolution", "energy", "force",
    "planet", "star", "chemical", "biology", "physics", "matter", "ecosystem",
    "light", "solar system", "electricity", "magnetism", "temperature", "velocity",
    "acceleration", "mass", "nucleus", "protein", "cell","force", "energy", "gravity", "velocity",
"inertia", "friction", "magnetism", "electricity", "voltage", "resistance","current", "circuit", "k
"refraction", "reflection", "laser", "sound", "atom", "particle", "electron","proton", "neutron", "
"entropy", "molecule", "compound", "element", "chemical", "reaction", "bond","ionic", "covalent", "
"inorganic", "periodic table", "oxidation", "polymer", "solution", "solvent",
"solute", "concentration", "equilibrium", "gas", "liquid", "solid", "combustion","distillation", "c
"metal", "nonmetal", "cell", "DNA", "gene", "chromosome", "photosynthesis",
"respiration", "bacteria", "virus", "evolution", "adaptation", "species","ecosystem", "biodiversity
"lipid", "organ", "tissue", "hormone", "neuron", "ecology", "population",
"organism", "metabolism", "mutation", "genetics", "inheritance", "biochemistry","zoology", "botany"
"tectonic", "earthquake", "volcano", "sediment", "erosion", "fossil","mineral", "rock", "magma", "c
"climate", "weather", "storm", "hurricane", "tornado", "precipitation",
"cloud", "water cycle", "ocean", "glacier", "tsunami", "flood", "drought",
"meteorology", "hydrology", "geosphere", "lithosphere", "stratosphere","troposphere", "ozone layer"
"universe", "galaxy", "star", "planet", "asteroid", "comet", "orbit","black hole", "nebula", "super
"satellite", "mars", "jupiter", "saturn", "neptune", "uranus", "mercury","venus", "milky way", "big
"red giant", "white dwarf", "climate change", "global warming", "greenhouse effect","carbon dioxide
"sustainability", "pollution", "deforestation", "recycling", "ozone layer","acid rain", "carbon foo
"hypothesis", "theory", "observation", "data", "research", "analysis","measurement", "discovery", "
"scientist", "equation", "variable", "constant", "simulation", "model","result", "conclusion", "met
"digestive system", "nervous system", "immune system", "endocrine system",
"hormone", "reproductive system", "DNA", "gene", "chromosome", "enzyme","antibody", "vaccine", "alg
"mean", "median", "mode", "graph", "measurement", "vector", "matrix","ratio", "derivative", "integr
"pi (π)"

    # we can add more keywords as we need to help with more classification
]

# Function to check if the question contains science related keywords
def contains_science_keyword(question):
    question = question.lower()
    for keyword in science_keywords:
        if keyword in question:
            return True
    return False
```

EXECUTION

STEP 5 - RESPONSE GENERATION

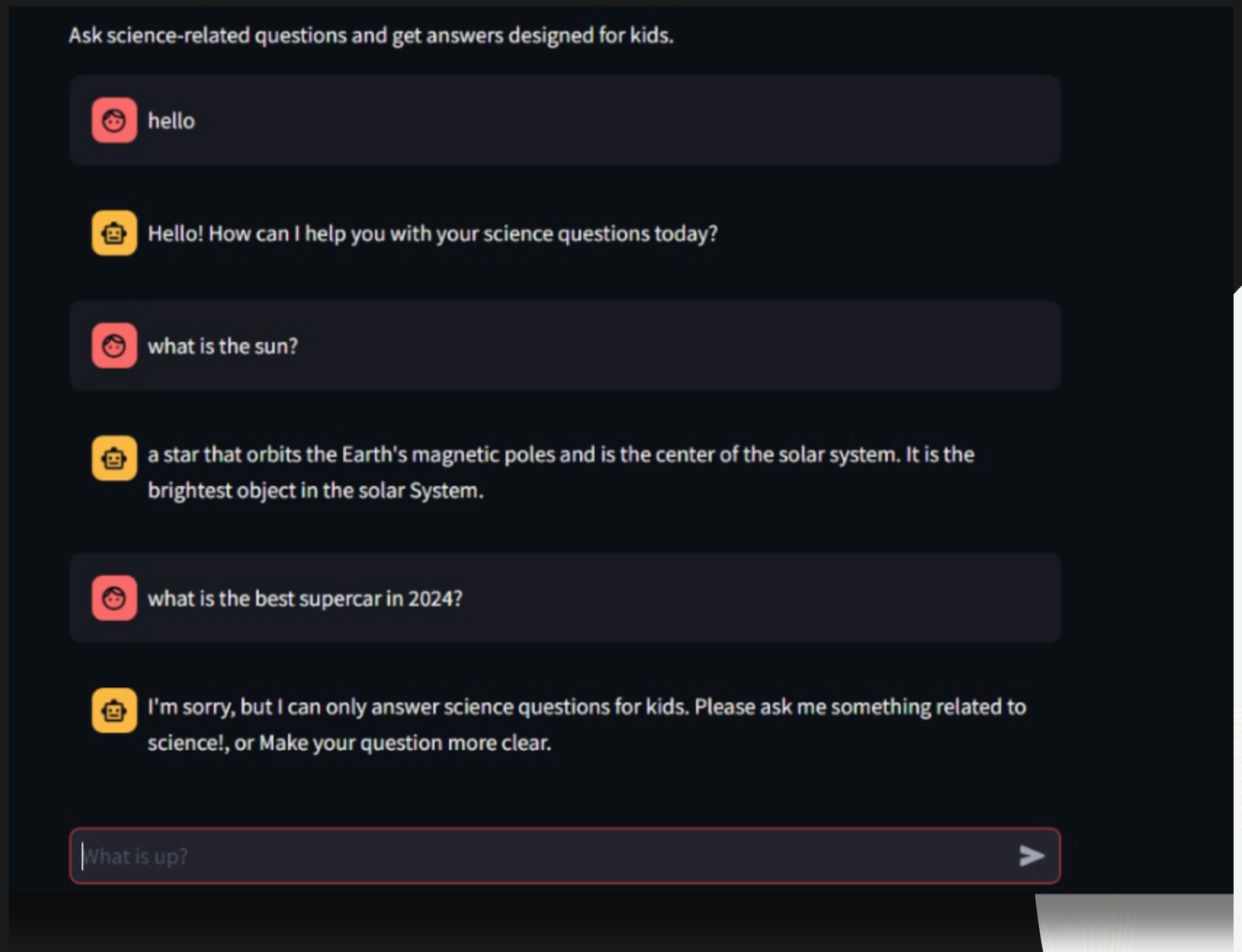


- We created a function to generate response
- Classify the answer before returning it

- Run the chatbot loop

```python
# the function to generate response
def generate_answer(question):
    if not is_science_question(question):
        return "I'm sorry, but I can only answer science questions for kids. Please ask me something related to

    # the input that will go to the generative model
    input_text = (
        f"You are a friendly and knowledgeable science teacher explaining concepts to kids. "
        f"Provide a detailed and engaging answer to the following science question.\n\n"
        f"Question: {question}\n\n"
        f"Answer:"
    )

    # we tokenize the input
    inputs = generator_tokenizer.encode(
        input_text,
        return_tensors='pt',
        max_length=512,
        truncation=True
    ).to(device)

    # then we  generate the answer
    with torch.no_grad():
        outputs = generator_model.generate(
            inputs,
            do_sample=True,
            max_length=300,
            min_length=30,
            temperature=0.5,
            top_p=0.9,
            repetition_penalty=1.2,
            no_repeat_ngram_size=3,
        )

    # decoding the answer before passing it
    answer = generator_tokenizer.decode(outputs[0], skip_special_tokens=True)

    # we remove the prompt from the response that will go to the user
    if answer.startswith(input_text):
        answer = answer[len(input_text):].strip()

    return answer.strip()

# then finally we implement chatbot loop
if __name__ == "__main__":
    print("Science Bot for Kids Chatbot is running. Type 'exit' to stop.")
    while True:
        user_input = input("You: ")
        if user_input.lower() in ["exit", "quit"]:
            print("Chatbot: Goodbye!")
            break

        # and here we generate the answer for the user
        response = generate_answer(user_input)

        print(f"Science Bot: {response}\n")
```


Kids Science ChatBot



ANALYSIS

OBSERVATION

- The Chatbot was able to reply to science-related questions and classify nonscience-related questions and decline to answer, while also being able to classify greetings and reply properly.

ANALYSIS

LIMITATIONS

- The Trec dataset is only 4500 questions which is not a large dataset and thus doesn't cover a large spectrum of science-related questions

- Training models is an expensive task requiring trial and error and enhancement iterations that require some investment to fine-tune the model properly.

- we used flan t5 large, as the model size won't need memory optimization on smaller GPUs

ENHANCEMENTS

- Utilizing a larger dataset will make the science bot more adequate and will not need another layer of keyword classifications.

- by using a larger model or utilizing an **API-based** model like GPT3.5 or GPT4 the bot performance and responses will be more natural and enhanced a lot with proper prompt engineering

OUR TEAM

Mohamed Hesham

Model Training & LLM Integration

Emad Mahmoud

UI & Integrations & Testing

Ahmed Mohamed

Dockerization & Testing

Kids Science Bot

THANKS FOR YOUR TIME.

Team *: Team X* :

Mohamed Hesham

Ahmed Mohamed

Emad Mahmoud

Course*: Gen-AI*

Group*: GIZ_AIS2_M1e*

Graduation Project *: Kids Science Bot*