**Second Iteration**

**Car Diagnostics Chatbot using BERT - Code Documentation**

|  |
| --- |
| **TensorFlow and Transformers Import:** |
| import tensorflow as tf import tensorflow\_hub as hub from transformers import BertTokenizer from transformers import TFBertForQuestionAnswering import numpy as np |
| This imports TensorFlow and TensorFlow Hub for deep learning functionalities, as well as necessary modules from the transformers library for using BERT models. |
|  |
| **Load Pre-trained BERT Model and Tokenizer:** |
| # Load pre-trained BERT model for question answering model = TFBertForQuestionAnswering.from\_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')  # Load tokenizer tokenizer = BertTokenizer.from\_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad') |
| This loads a pre-trained BERT model fine-tuned for question answering and its corresponding tokenizer. |
|  |
| **Predefined Question-Answer Pairs:** |
| # Dictionary containing predefined questions and answers qa\_pairs = {  "What should I do if my check engine light is on?": "If your check engine light is on, it could indicate a variety of issues. It's best to have your vehicle diagnosed by a professional mechanic to determine the specific problem.",  "How often should I change my oil?": "The frequency of oil changes depends on your vehicle's make, model, and driving conditions. Typically, it's recommended to change your oil every 5,000 to 7,500 miles, but refer to your owner's manual for specific guidelines.",  "What does it mean if my car is making a strange noise?": "Strange noises could indicate various problems with your vehicle, such as worn-out brakes, a faulty belt, or a problem with the transmission. It's important to have these noises inspected by a mechanic as soon as possible.",  "How can I improve my car's fuel efficiency?": "To improve fuel efficiency, you can ensure proper tire inflation, regular maintenance, avoid aggressive driving, and reduce unnecessary weight in your vehicle.",  "What should I do if my car is overheating?": "If your car is overheating, pull over to a safe location immediately and turn off the engine. Allow the engine to cool down before attempting to open the hood. Check the coolant level and radiator hoses for any leaks or damage. If you're unable to resolve the issue, seek assistance from a professional mechanic." } |
| This dictionary contains predefined questions as keys and their corresponding answers as values. |
|  |
| **Function for Question Answering:** |
| # Function to perform question answering def answer\_question(question, context):  if question in qa\_pairs:  return qa\_pairs[question]  else:  # Tokenize inputs  input\_ids = tokenizer.encode(question, context)  token\_type\_ids = [0 if i <= input\_ids.index(102) else 1 for i in range(len(input\_ids))]  input\_ids = np.array([input\_ids])  token\_type\_ids = np.array([token\_type\_ids])   # Get model prediction  start\_scores, end\_scores = model.predict([input\_ids, token\_type\_ids])   # Find the tokens with the highest start and end scores  answer\_start = np.argmax(start\_scores)  answer\_end = np.argmax(end\_scores)   # Get the answer span  answer = tokenizer.convert\_tokens\_to\_string(tokenizer.convert\_ids\_to\_tokens(input\_ids[0][answer\_start:answer\_end+1]))   return answer |
| This function takes a question and context as input and returns an answer. If the question is predefined, it returns the answer from qa\_pairs, otherwise, it uses the BERT model to predict the answer. |
|  |
| **Conversation Loop:** |
| # Example conversation loop while True:  question = input("Customer: ")  if question.lower() in ['exit', 'quit', 'bye']:  print("Chatbot: Goodbye!")  break  context = "Car diagnostics context here..." # Provide relevant car diagnostics information here  answer = answer\_question(question, context) |
| This sets up a loop where the user can input questions. The loop continues until the user inputs 'exit', 'quit', or 'bye'. |
|  |
| **Handling User Input:**  If the user inputs 'exit', 'quit', or 'bye', the loop exits and the program ends.  Otherwise, it retrieves the context (which is assumed to be related to car diagnostics) and calls answer\_question() to get the response.  **Output the Answer:** |
| print("Chatbot:", answer) |

**Bug #1**

code currently only checks if the question is exactly the same as the predefined questions in the qa\_pairs dictionary. Since the question "My check engine light is on, what should I do?" is not exactly the same as any of the predefined questions, the code falls back to using BERT for question answering, which might not provide a satisfactory answer for this specific question.

**Fix #1**

dictionary, allowing for some flexibility in matching. One way to achieve this is by using a similarity measure

|  |
| --- |
| **Import SequenceMatcher from difflib module:** |
| from difflib import SequenceMatcher |
| This import statement brings in the SequenceMatcher class from the difflib module, which will be used to calculate the similarity between two strings. |
|  |
| **Add similarity function:** |
| def similarity(a, b):  return SequenceMatcher(None, a, b).ratio() |
| This function takes two strings a and b as input and calculates the similarity between them using SequenceMatcher. It returns a similarity score between 0 and 1, where 1 indicates exact similarity. |
|  |
| **Modify answer\_question function:** |
| def answer\_question(question, context):  # Preprocess input question  question = question.lower()   # Check if the question matches any predefined question with some level of similarity  max\_similarity = 0  best\_match = None  for q in qa\_pairs.keys():  sim = similarity(question, q.lower())  if sim > max\_similarity:  max\_similarity = sim  best\_match = q   # If a match with sufficient similarity is found, return the corresponding answer  if max\_similarity >= 0.7: # Adjust the threshold as needed  return qa\_pairs[best\_match]  else:  # Tokenize inputs  input\_ids = tokenizer.encode(question, context)  token\_type\_ids = [0 if i <= input\_ids.index(102) else 1 for i in range(len(input\_ids))]  input\_ids = np.array([input\_ids])  token\_type\_ids = np.array([token\_type\_ids])   # Get model prediction  start\_scores, end\_scores = model.predict([input\_ids, token\_type\_ids])   # Find the tokens with the highest start and end scores  answer\_start = np.argmax(start\_scores)  answer\_end = np.argmax(end\_scores)   # Get the answer span  answer = tokenizer.convert\_tokens\_to\_string(tokenizer.convert\_ids\_to\_tokens(input\_ids[0][answer\_start:answer\_end+1]))   return answer |
|  |
| The answer\_question function now preprocesses the input question to lowercase before calculating its similarity with predefined questions.  It iterates through the predefined questions, calculates the similarity between the input question and each predefined question, and selects the best match based on similarity.  If the best match has a similarity score above a threshold (0.7 in this case), it returns the corresponding answer from the qa\_pairs dictionary.  If no sufficiently similar match is found, the function falls back to using BERT for question answering as before. |

**Added some rudimentary questions and answers**

"What should I do if my check engine light is on?": "If your check engine light is on, it could indicate a variety of issues. It's best to have your vehicle diagnosed by a professional mechanic to determine the specific problem.",  
"How often should I change my oil?": "The frequency of oil changes depends on your vehicle's make, model, and driving conditions. Typically, it's recommended to change your oil every 5,000 to 7,500 miles, but refer to your owner's manual for specific guidelines.",  
"What does it mean if my car is making a strange noise?": "Strange noises could indicate various problems with your vehicle, such as worn-out brakes, a faulty belt, or a problem with the transmission. It's important to have these noises inspected by a mechanic as soon as possible.",  
"How can I improve my car's fuel efficiency?": "To improve fuel efficiency, you can ensure proper tire inflation, regular maintenance, avoid aggressive driving, and reduce unnecessary weight in your vehicle.",  
"What should I do if my car is overheating?": "If your car is overheating, pull over to a safe location immediately and turn off the engine. Allow the engine to cool down before attempting to open the hood. Check the coolant level and radiator hoses for any leaks or damage. If you're unable to resolve the issue, seek assistance from a professional mechanic.",  
"What could be the cause if my car won't start?": "If your car won't start, it could be due to a dead battery, faulty starter motor, or issues with the ignition system. Check the battery connections, try jump-starting the vehicle, and ensure the key is in the correct position before seeking further assistance.",  
"Why is my steering wheel vibrating?": "Vibrations in the steering wheel could be caused by unbalanced tires, worn-out suspension components, or problems with the wheel bearings. Have your tires balanced and rotated regularly, and inspect the suspension system for any signs of wear or damage.",  
"What should I do if my brakes feel spongy?": "Spongy brakes could indicate air in the brake lines, low brake fluid levels, or worn-out brake pads. Check the brake fluid reservoir for proper levels and inspect the brake lines for any signs of leaks. Bleeding the brake system may also help remove any air bubbles.",  
"What does it mean if my car is pulling to one side?": "Pulling to one side could be caused by misaligned wheels, uneven tire pressure, or worn-out suspension components. Have your wheel alignment checked and ensure that tires are inflated to the recommended pressure. Inspect the suspension system for any signs of damage or wear.",  
"Why is my engine misfiring?": "Engine misfires could be caused by faulty spark plugs, a clogged fuel injector, or issues with the ignition system. Check the spark plugs for signs of wear and replace them if necessary. Inspect the fuel injectors for any obstructions and consider cleaning or replacing them if needed.",  
"What should I do if my transmission is slipping?": "A slipping transmission could indicate low transmission fluid levels, worn-out clutch plates, or issues with the transmission solenoid. Check the transmission fluid level and condition, and top it up if necessary. If the problem persists, have the transmission inspected by a professional mechanic.",  
"How do I know if my alternator is failing?": "Signs of a failing alternator include dimming headlights, a dead battery, or warning lights on the dashboard. Use a multimeter to test the alternator output voltage, and have it inspected by a mechanic if you suspect it's failing.",  
"Why is my exhaust emitting smoke?": "Smoke from the exhaust could indicate various issues, such as burning oil, coolant leaks, or a rich fuel mixture. Check the oil and coolant levels for any signs of contamination or leaks. If the smoke persists, have the exhaust system inspected by a professional mechanic.",  
"What should I do if my air conditioning is blowing hot air?": "Hot air from the air conditioning system could be caused by low refrigerant levels, a faulty compressor, or issues with the cooling fans. Check the refrigerant levels and ensure that the compressor is engaging properly. If the problem persists, have the air conditioning system inspected by a professional.",  
"Why is my car's suspension squeaking?": "Squeaking noises from the suspension could indicate worn-out bushings, damaged shock absorbers, or issues with the suspension mounts. Inspect the suspension components for any signs of wear or damage, and lubricate the bushings if necessary. If the squeaking persists, have the suspension system inspected by a professional mechanic.",  
"What should I do if my windshield wipers are streaking?": "Streaking windshield wipers could be caused by worn-out wiper blades, a dirty windshield, or issues with the wiper arms. Replace the wiper blades if they're worn or damaged, and clean the windshield thoroughly. If the streaking persists, inspect the wiper arms for any signs of damage or misalignment.",  
"How do I know if my wheel bearings are failing?": "Signs of failing wheel bearings include grinding or humming noises coming from the wheels, uneven tire wear, or excessive play in the wheels. Jack up the vehicle and check for any excessive play or roughness in the wheel bearings. If you suspect they're failing, have them replaced by a professional mechanic.",  
"What should I do if my car is stalling?": "Stalling could be caused by various issues, such as a faulty fuel pump, clogged fuel filter, or problems with the ignition system. Check the fuel pump and filter for proper operation and replace them if necessary. Inspect the ignition system components for any signs of wear or damage.",  
"Why is my car's airbag light on?": "An illuminated airbag light could indicate a problem with the airbag system, such as a faulty sensor or wiring issue. Have the airbag system inspected by a professional mechanic to diagnose the specific problem and ensure that the airbags are functioning properly.",  
"What should I do if my car is vibrating at high speeds?": "Vibrations at high speeds could be caused by unbalanced tires, a bent wheel, or problems with the suspension system. Have your tires balanced and rotated regularly, and inspect the wheels for any signs of damage or deformation. If the vibrations persist, have the suspension system inspected by a professional mechanic.",  
"How do I know if my catalytic converter is failing?": "Signs of a failing catalytic converter include reduced engine performance, sulfuric odors from the exhaust, or illuminated warning lights on the dashboard. Have the exhaust system inspected for any signs of damage or clogging, and replace the catalytic converter if necessary.",  
"What should I do if my power steering is making noise?": "Noises from the power steering system could indicate low power steering fluid levels, a worn-out power steering pump, or issues with the steering rack. Check the power steering fluid reservoir for proper levels and inspect the system for any signs of leaks. If the noise persists, have the power steering system inspected by a professional mechanic.",  
"Why is my car's battery draining quickly?": "A quickly draining battery could be caused by a faulty altern"

**Output**

**Custome**r: my suspensions are squeaking ?

**Chatbot**: Squeaking noises from the suspension could indicate worn-out bushings, damaged shock absorbers, or issues with the suspension mounts. Inspect the suspension components for any signs of wear or damage, and lubricate the bushings if necessary. If the squeaking persists, have the suspension system inspected by a professional mechanic.

**Changed The Premade model**

**From**

# Load pre-trained BERT model for question answering  
model = TFBertForQuestionAnswering.from\_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')  
  
# Load tokenizer  
tokenizer = BertTokenizer.from\_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')

**To**

model = TFBertForQuestionAnswering.from\_pretrained('bert-large-uncased')  
  
# Load tokenizer  
tokenizer = BertTokenizer.from\_pretrained('bert-large-uncased')

**THIRD ITTERATIONS**

**CSV File Loading**

import pandas as pd  
  
qa\_pairs = pd.read\_csv('vehicle\_dataset.csv').set\_index('Vehicle').T.to\_dict()

**Explanation**: The new approach uses pandas to directly read the CSV file into a DataFrame, which is then converted to a dictionary. This simplifies the code and handles the CSV parsing more efficiently.

**Answer Question Function**

def answer\_question(question):  
 # Preprocess input question  
 question = question.lower()  
  
 # Check if the question matches any predefined question with some level of similarity  
 max\_similarity = 0  
 best\_match = None  
 for q in qa\_pairs.keys():  
 sim = similarity(question, q)  
 if sim > max\_similarity:  
 max\_similarity = sim  
 best\_match = q  
  
 # If a match with sufficient similarity is found, construct and return the response  
 if max\_similarity >= 0.4:  
 vehicle\_data = qa\_pairs[best\_match]  
 part\_match = re.search(r'\b(oil filter|air filter|spark plugs|windshield wipers|brake pads|battery)\b', question)  
 if part\_match:  
 part = part\_match.group(1)  
 return f"The replacement {part} for {best\_match.title()} is {vehicle\_data[part]}. Here are the steps for installation: {vehicle\_data[f'{part} instructions']}"  
 elif 'toyota corolla 2018' in question and 'spark plug' in question:  
 return f"The replacement spark plugs for Toyota Corolla 2018 are {vehicle\_data['spark plug']}. Here are the steps for installation: {vehicle\_data['spark plug instructions']}"  
 else:  
 return "I'm sorry, I couldn't find an answer to your question."  
 else:  
 return "I'm sorry, I couldn't find an answer to your question."

**Explanation**: The new answer question function includes preprocessing of the input question, matching with predefined questions, and constructing the response based on the matched question. It uses regular expressions for pattern matching and constructs responses for specific parts or vehicle models.

**Fourth Iteration**

**Importing Libraries**

import csv  
from transformers import BertTokenizer, TFBertForQuestionAnswering  
import numpy as np  
from difflib import SequenceMatcher

**Explanation:**

csv: Used for reading and writing CSV files.

transformers.BertTokenizer and transformers.TFBertForQuestionAnswering: Used for BERT model and tokenizer.

numpy: Used for numerical operations.

difflib.SequenceMatcher: Used for comparing similarity between strings.

**Loading Pre-trained BERT Model and Tokenizer**

# Load pre-trained BERT model for question answering  
model = TFBertForQuestionAnswering.from\_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')  
  
# Load tokenizer  
tokenizer = BertTokenizer.from\_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')

**Explanation:**

TFBertForQuestionAnswering.from\_pretrained(): Loads a pre-trained BERT model for question answering.

BertTokenizer.from\_pretrained(): Loads a pre-trained BERT tokenizer.

**Function to Calculate Similarity Between Strings**

def similarity(a, b):  
 return SequenceMatcher(None, a, b).ratio()

**Explanation:**

SequenceMatcher: Compares two strings and returns a similarity ratio.

def read\_qa\_pairs\_from\_csv(csv\_file):  
 qa\_pairs = {}  
 with open(csv\_file, mode='r') as file:  
 csv\_reader = csv.reader(file)  
 for row in csv\_reader:  
 qa\_pairs[row[0]] = row[1]  
 return qa\_pairs

**Explanation:**

open(csv\_file, mode='r'): Opens the CSV file in read mode.

csv.reader(file): Creates a CSV reader object to read the file.

qa\_pairs[row[0]] = row[1]: Adds each question-answer pair to a dictionary.

**Function to Perform Question Answering**

def answer\_question(question, context, qa\_csv\_file):  
 # Load QA pairs from the CSV file  
 qa\_pairs = read\_qa\_pairs\_from\_csv(qa\_csv\_file)  
  
 # Preprocess input question  
 question = question.lower()  
  
 # Check if the question matches any predefined question with some level of similarity  
 max\_similarity = 0  
 best\_match = None  
 for q in qa\_pairs.keys():  
 sim = similarity(question, q.lower())  
 if sim > max\_similarity:  
 max\_similarity = sim  
 best\_match = q  
  
 # If a match with sufficient similarity is found, return the corresponding answer  
 if max\_similarity >= 0.7: # Adjust the threshold as needed  
 return qa\_pairs[best\_match]  
 else:  
 # Tokenize inputs  
 input\_ids = tokenizer.encode(question, context)  
 token\_type\_ids = [0 if i <= input\_ids.index(102) else 1 for i in range(len(input\_ids))]  
 input\_ids = np.array([input\_ids])  
 token\_type\_ids = np.array([token\_type\_ids])  
  
 # Get model prediction  
 start\_scores, end\_scores = model.predict([input\_ids, token\_type\_ids])  
  
 # Find the tokens with the highest start and end scores  
 answer\_start = np.argmax(start\_scores)  
 answer\_end = np.argmax(end\_scores)  
  
 # Get the answer span without the [CLS] token  
 answer = tokenizer.convert\_tokens\_to\_string(tokenizer.convert\_ids\_to\_tokens(input\_ids[0][answer\_start+1:answer\_end+1]))  
  
 return answer

**Explanation:**

read\_qa\_pairs\_from\_csv(qa\_csv\_file): Reads the question-answer pairs from the CSV file.

question.lower(): Converts the input question to lowercase for case-insensitive matching.

tokenizer.encode(question, context): Tokenizes the question and context using the BERT tokenizer.

model.predict([input\_ids, token\_type\_ids]): Makes a prediction using the BERT model.

tokenizer.convert\_ids\_to\_tokens(): Converts token IDs back to tokens.

The function returns the answer to the question.

**FIFTH ITTERATION**

**Importing Libraries:**

from flask import Flask, request, jsonify  
from flask\_cors import CORS # Import CORS from flask\_cors  
import csv  
from transformers import BertTokenizer, TFBertForQuestionAnswering  
import numpy as np  
from difflib import SequenceMatcher

Flask: A web framework for building web applications in Python.

request: Allows handling HTTP requests in Flask.

jsonify: Converts Python dictionaries to JSON format.

CORS: Cross-Origin Resource Sharing middleware for handling cross-origin requests.

csv: Library for reading and writing CSV files.

BertTokenizer and TFBertForQuestionAnswering: Components from the transformers library for using BERT models for question answering.

numpy: Library for numerical computations in Python.

SequenceMatcher: A class from the difflib module used to calculate string similarity.

Initializing Flask App and CORS:

python

app = Flask(\_\_name\_\_)  
CORS(app) # Apply CORS to your Flask app

**Creates a Flask application instance named app.**

Applies CORS (Cross-Origin Resource Sharing) to allow cross-origin requests.

Loading Pre-Trained BERT Model and Tokenizer:

python

# Load pre-trained BERT model for question answering  
model = TFBertForQuestionAnswering.from\_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')  
  
# Load tokenizer  
tokenizer = BertTokenizer.from\_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')

Loads a pre-trained BERT model for question answering (TFBertForQuestionAnswering) using the Hugging Face Transformers library.

Loads the corresponding tokenizer (BertTokenizer) for tokenizing input text.

**Similarity Function:**

python

def similarity(a, b):  
 return SequenceMatcher(None, a, b).ratio()

Defines a function similarity that calculates the similarity ratio between two strings using SequenceMatcher.

**Reading QA Pairs from CSV:**

python

def read\_qa\_pairs\_from\_csv(csv\_file):  
 qa\_pairs = {}  
 with open(csv\_file, mode='r', newline='') as file:  
 csv\_reader = csv.reader(file)  
 for row in csv\_reader:  
 if len(row) >= 2: # Check if the row has at least two elements (question and answer)  
 qa\_pairs[row[0]] = row[1]  
 return qa\_pairs

Defines a function read\_qa\_pairs\_from\_csv to read question-answer pairs from a CSV file and store them in a dictionary.

**Answering Question Function:**

python

def answer\_question(question, context, qa\_pairs):  
 # Preprocess input question  
 question = question.lower()  
  
 # Check if the question matches any predefined question with some level of similarity  
 max\_similarity = 0  
 best\_match = None  
 for q in qa\_pairs.keys():  
 sim = similarity(question, q.lower())  
 if sim > max\_similarity:  
 max\_similarity = sim  
 best\_match = q  
  
 # If a match with sufficient similarity is found, return the corresponding answer  
 if max\_similarity >= 0.7: # Adjust the threshold as needed  
 return qa\_pairs[best\_match]  
 else:  
 # Tokenize inputs  
 input\_ids = tokenizer.encode(question, context)  
 token\_type\_ids = [0 if i <= input\_ids.index(102) else 1 for i in range(len(input\_ids))]  
 input\_ids = np.array([input\_ids])  
 token\_type\_ids = np.array([token\_type\_ids])  
  
 # Get model prediction  
 start\_scores, end\_scores = model.predict([input\_ids, token\_type\_ids])  
  
 # Find the tokens with the highest start and end scores  
 answer\_start = np.argmax(start\_scores)  
 answer\_end = np.argmax(end\_scores)  
  
 # Get the answer span without the [CLS] token  
 answer = tokenizer.convert\_tokens\_to\_string(tokenizer.convert\_ids\_to\_tokens(input\_ids[0][answer\_start+1:answer\_end+1]))  
  
 return answer

Defines a function answer\_question to answer a given question using either the pre-defined QA pairs or the BERT model, depending on the similarity of the question to the predefined questions.

**Flask Route for Handling POST Requests:**

python

app.route('/ask', methods=['POST'])  
def ask():  
 if request.method == 'POST':  
 data = request.get\_json()  
 question = data.get('question')  
 context = "Car diagnostics context here..." # Provide relevant car diagnostics information here  
 qa\_csv\_file = 'dataset.csv'  
 qa\_pairs = read\_qa\_pairs\_from\_csv(qa\_csv\_file)  
 answer = answer\_question(question, context, qa\_pairs)  
 return jsonify({'answer': answer})  
 else:  
 return jsonify({'error': 'Method not allowed'}), 405

Creates a Flask route /ask that accepts POST requests.

Retrieves the question from the JSON data sent in the POST request.

Defines the context for answering questions (you'll need to provide relevant information here).

Reads QA pairs from a CSV file using read\_qa\_pairs\_from\_csv.

Calls answer\_question to get the answer and returns it as JSON.

**Running the Flask App:**

python

if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(port=5000)

Runs the Flask application on port 5000 when the script is executed directly.

This code sets up a Flask API for handling questions about car diagnostics. It uses a CSV file (dataset.csv) containing predefined question-answer pairs and a BERT model for question answering. When a question is posted to the /ask endpoint, the API returns the answer based on the pre-defined QA pairs or the BERT model's prediction.

**Model Evaluation :**

Model: bert-base-uncased  
BERTScore Precision: 0.8372  
BERTScore Recall: 0.8939  
BERTScore F1 score: 0.8646

Model: bert-large-uncased  
BERTScore Precision: 0.8526  
BERTScore Recall: 0.9306  
BERTScore F1 score: 0.8899

Model: bert-base-cased  
BERTScore Precision: 0.9410  
BERTScore Recall: 0.9467  
BERTScore F1 score: 0.9438

Model: bert-large-cased  
BERTScore Precision: 0.7528  
BERTScore Recall: 0.7779  
BERTScore F1 score: 0.7651

Model: bert-base-multilingual-uncased  
BERTScore Precision: 0.8055  
BERTScore Recall: 0.8215  
BERTScore F1 score: 0.8134  
  
Model: bert-base-multilingual-cased  
BERTScore Precision: 0.8754  
BERTScore Recall: 0.9460  
BERTScore F1 score: 0.9094

Model: bert-large-uncased-whole-word-masking-finetuned-squad  
BERTScore Precision: 0.7814  
BERTScore Recall: 0.7884  
BERTScore F1 score: 0.7849

**A graph with blue bars

Description automatically generated with medium confidence**

Based on the BERTScore evaluation results for the pre-trained BERT models, the most suitable model depends on the specific criteria you prioritize. Here is a summary of the findings:

BERT-base-cased:

BERTScore Precision: 0.9410

BERTScore Recall: 0.9467

BERTScore F1 score: 0.9438

This model stands out with the highest BERTScore F1 score among the models listed. It achieves a good balance between precision and recall, indicating robust performance in a question-answering task.

However, it's essential to consider other factors such as model size, computational resources required for inference, and the specific nature of your application. For instance, larger models like BERT-large-uncased may offer slightly better recall but come with increased computational demands.

Therefore, the best-suited BERT model depends on your project's requirements, including performance metrics, computational constraints, and the specific tasks the model needs to excel in.

**Objectives and Goals of Testing:**

The objectives and goals of testing for the chatbot project are:

The objectives and goals of testing for the chatbot project are multifaceted. Firstly, the testing aims to ensure that the chatbot provides accurate recommendations and retrieves information correctly regarding vehicle spare parts, enhancing user experience and utility. Secondly, it focuses on validating the correctness and clarity of application instructions, ensuring users receive clear and accurate guidance on tasks like part replacement. Thirdly, the testing verifies the accuracy of answers and knowledge delivery for general queries, maintaining the chatbot's reliability as an information source. Lastly, the testing evaluates the performance, scalability, and security of the chatbot system, ensuring it can handle varying loads, maintain responsiveness, and protect user data effectively.

**Testing Criteria:**

**Accuracy of Spare Part Recommendations and Application Instructions**

The chatbot is designed to provide accurate recommendations for vehicle spare parts and clear instructions for their application. To ensure accuracy, the chatbot leverages a pre-trained BERT model for natural language understanding and retrieval of relevant information from a dataset containing spare part details and application instructions. The accuracy of spare part recommendations and application instructions is evaluated based on the following criteria:

* Matching recommended spare parts with user queries.
* Providing correct and detailed application instructions for recommended parts.

**Precision in Answering General Knowledge Queries**

In addition to spare part recommendations, the chatbot is capable of answering general knowledge queries related to vehicles, maintenance procedures, and automotive industry topics. Precision in answering these queries is crucial for providing reliable information to users. The chatbot's precision is evaluated based on:

* Correctness and relevance of answers to general knowledge queries.
* Clarity and conciseness of explanations provided.

**Response Time and Performance Metrics During Peak Usage**

During peak usage periods, the chatbot's response time and overall performance are critical factors. The chatbot's response time is measured from the moment a user query is received to the delivery of the response. Performance metrics include:

* Average response time under normal and peak loads.
* Concurrent user handling capacity without degradation in response time.
* System resource utilization (CPU, memory) during peak usage.

**Scalability to Handle Increasing User Load Without Degradation**

Scalability is essential for ensuring that the chatbot can handle an increasing number of users without compromising performance or response time. The chatbot's scalability is assessed based on:

* Ability to scale horizontally (adding more servers) or vertically (increasing server capacity) to meet demand.
* Load balancing mechanisms to distribute user queries effectively across servers.
* Performance testing under simulated high-traffic scenarios to validate scalability.

**Security Protocols for Data Privacy and Protection Against Malicious Inputs**

To safeguard user data and prevent malicious inputs, the chatbot implements robust security protocols. These protocols include:

* **Encryption of Sensitive Data**
  + Sensitive data, such as user information and communication, is encrypted using industry-standard encryption protocols during both transmission and storage. This ensures that data remains secure and protected from unauthorized access or interception.
* **Input Validation for Malicious Inputs**
  + The chatbot employs robust input validation techniques to detect and mitigate malicious inputs, including common vulnerabilities like SQL injection and cross-site scripting (XSS). By validating user inputs at various levels, the system prevents potential security breaches and data manipulation attempts.
* **Access Controls and Authentication Mechanisms**
  + Access to the chatbot system is strictly controlled through comprehensive access controls and authentication mechanisms. Only authorized users with valid credentials can interact with the system, reducing the risk of unauthorized access and ensuring data privacy.
* **Regular Security Audits and Updates**
  + The chatbot project undergoes regular security audits and updates to proactively address emerging threats and vulnerabilities. This includes reviewing system configurations, applying security patches, and implementing best practices to enhance overall security posture and resilience against cyber threats.

**Functional Testing:**

Functional testing ensures that the chatbot meets its intended functional requirements. This includes:

* **Testing Spare Part Recommendation Accuracy**
  + The accuracy of spare part recommendations is tested by evaluating the chatbot's ability to match user queries with the correct spare parts from the dataset. This involves inputting various queries related to vehicle parts and assessing whether the chatbot recommends the appropriate parts based on the query context.
* **Verifying Correctness of Application Instructions**
  + Application instructions provided by the chatbot are verified for correctness by comparing them against established guidelines or manuals. Test cases are designed to cover different scenarios, ensuring that the instructions accurately reflect the recommended procedures for installing or using the recommended spare parts.
* **Validating General Knowledge Responses**
  + General knowledge responses are validated by posing queries on automotive topics or maintenance procedures and evaluating the accuracy and relevance of the chatbot's answers. This validation process includes assessing the clarity and completeness of the information provided in response to diverse queries.
* **Ensuring Seamless Conversation Flows**
  + The chatbot's conversation flows are tested to ensure smooth transitions between user queries and responses. This involves testing the chatbot's ability to handle follow-up questions, clarify user intents, and maintain context throughout the conversation. The goal is to provide users with a seamless and intuitive conversational experience.

**Module and Integration Testing:**

Module and integration testing focus on testing individual components and their integration within the chatbot system. This includes:

* Testing individual recommendation algorithms for spare parts.
* Validating the accuracy of knowledge retrieval modules.
* Ensuring smooth integration of various modules for coherent responses.

**Accuracy Testing:**

* Checking the accuracy of spare part recommendations and application instructions.
* Verifying the correctness of general knowledge responses.

**Performance Testing:**

* Assessing response times and system performance under varying loads.
* Testing scalability to handle increasing user traffic.

**Load Balance and Scalability:**

Testing the system's ability to balance loads and scale resources as needed.

Evaluating performance during peak usage periods.

Security Testing:

Checking for vulnerabilities and ensuring data privacy and protection.

Testing input validation and handling of malicious inputs.

**Limitations of the Testing Process:**

* Testing may not cover all possible user scenarios.
* Real-world user interactions may vary from test scenarios.
* Limitations in testing scalability to extreme load conditions.