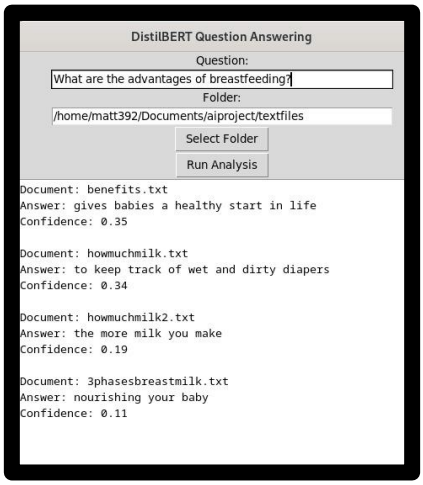


```
"""
Flowchart of program:
>Import tools and modules from Python
> Set up cache directory
> Load AI model: DistilBERT
> Load the correct tokenizer
> Specify task: question and answer pipeline
> Start function to extract data from files
> For each text document in folder:
    > Extract information from text
    > Process through DistilBERT
    > Get score from DistilBERT
    > If score above threshold, add to results
> Specify question and document folder
> Sort results by score
> Print results through Tkinter GUI
"""
```

```python
# The os library interacts with the operating system
import os
# Transformers library, developed by Hugging Face, is for natural language
processing
# pipeline is a function in the transformers library;
# makes use of the pre-trained models simpler
# AutoTokenizer automatically selects and loads the appropriate tokenizer
# Tokenization converts the raw text into format that can be understood by
machine learning models
# AutoModelForQuestionAnswering class selects the correct architecture based
on the model being used
from transformers import pipeline, AutoTokenizer, AutoModelForQuestionAnswering
# itertools is a Python module provides tools for working with iterators
# islice function extracts a portion of data ("slices it")
from itertools import islice
# os.environ represents environment variables in the current process
# Can read or set variables
# TRANSFORMERS_CACHE is where pre-trained models are stored
os.environ['TRANSFORMERS_CACHE'] = '/home/matt392/transformers_cache'
```

```
# GUI related imports
# tkinter provides tools for graphical user interfaces
# imported as "tk"
import tkinter as tk
# filedialog is a submodulte that provides for file selection
from tkinter import filedialog

# Set up cache directory
cache_dir = '/home/matt392/transformers_cache'
# Creates a directory at the path specified in "cache_dir"
# If it exists already, it will not throw an error
os.makedirs(cache_dir, exist_ok=True)
# The information in variable "cache_dir" is placed into
# the environment variable "TRANSFORMERS_CACHE"
os.environ['TRANSFORMERS_CACHE'] = cache_dir
# Print the location of the cache directory defined previously
print(f"Using cache directory: {cache_dir}")
```

```
# The following code will load the small DistilBERT Question and Answer model
# It was fine tuned on SQuAD (Stanford Question Answering Dataset).
########
# Assigns string identifier to the variable "model_name"
model_name = "distilbert-base-uncased-distilled-squad"
# Loads and initializes the correct tokenizer associated with the AI model
# The tokenizer preprocesses the text; is separate from the actual model
tokenizer = AutoTokenizer.from_pretrained(model_name)
# Loads and initializes actual model
# Assigns to variable "model"
model = AutoModelForQuestionAnswering.from_pretrained(model_name)
# Utilizes the pipeline function to specify the task type ("question and
answer")
# Uses the model and tokenzier
# Assigns to variable "qa_pipeline" which can be used for question/answering
tasks
qa_pipeline = pipeline("question-answering", model=model, tokenizer=tokenizer)
# Confirms that the model has been loaded without errors
print(f"Model {model_name} loaded successfully")
```

```
# Function to extract information from documents
# Parameters are "documents_folder" and "question"
# "question" is the query that will determine what information is extracted
def extract_breastfeeding_info(documents_folder, question):
# create empty list called "results"
    results = []

    print("Starting document processing...")

# List documents in folder and place into "document" variable on each iteration
# Using "listdir" function in "os" module
    for document in os.listdir(documents_folder):
    # Print document names that are .txt files
        if document.endswith(".txt"):
            print(f"Processing file: {document}")
```

```
# Start "try" block
    try:
    # Join the path of the "documents_folder" with "document" to
    create full path
    # Open the document under "read" mode
    # Assign open file object to name "file"
        with open(os.path.join(documents_folder, document), "r") as
        file:
            # Read the contents of the file and place into variable
            "context"
            context = file.read()
            # Use "strip" function to remove whitespaces
            # if the file is empty, print a warning
            if not context.strip():
                print(f"Warning: File {document} is empty!")
            # If the file contains text, print out the number of
            characters using the "len" function
            else:
                print(f"File {document} contains text. Length: {len(
                context)} characters")
    # Catch any exceptions that happen when reading the file
    except Exception as e:
        # Print the error message
        print(f"Error reading file {document}: {e}")
        # Move on to the next file in the "for" loop
        continue


### HEART OF THE PROGRAM IS THE NEXT LINE BELOW ###
# Use "qa_pipeline" function created earlier takes a dictionary
as an argument
# With "question" and "context" keys
# These keys are processed through DistilBERT
# Result is a dictionary stored in variable "answer_dictionary"
answer_dictionary = qa_pipeline({"question": question, "context":
context})
```

```
    # Take the key "answer" from the "answer_dictionary" and place it
    # into variable "answer_text"
    answer_text = answer_dictionary["answer"]
    # Take the key "score" from "answer_dictionary" and
    # place it into variable "score"
    score = answer_dictionary["score"]
    # DistilBERT's confidence score can go from 0 to 1
    # higher is more confident
    # Higher the number, the stricter the filtering
    # Current score is .1; change confidence threshold as needed
    if score > 0.1:
        # If the score is above the threshold add tuple with 3 elements
        # to "results" list
        # Elements are:
        # Name of the document, extracted text, and confidence score
        results.append((document, answer_text, score))
    # If DistilBERT produces a score that is lower than the threshold
    # it will print the low score, the question and
    # the document that it reviewed
    else:
        print(f"Low confidence answer (score: {score}) for question:{
        question} in document: {document}")

    # Using the "listdir" function of the "os" module, list *all* of the
    files in the folder
    # not just the files that were processed
    # The "len" function will count the number of files in the list
    print(f"\nProcessed {len(os.listdir(documents_folder))} files.\n")

    # Send the "results" list back to the point where the function was called
    # The list has the tuples (document name, answer text, confidence score)
    # for the documents that meet or exceed the confidence threshold
    return results
# end function



# Graphical user interface code is below

# Using the "filedialog" submodule which will allow user to choose
# folder by running the "askdirectory()" function

# This block only runs if the script is executed directly (not imported)
if __name__ == "__main__":
    # Define select_folder function will allow user to select a folder and
    update the GUI
    def select_folder():
        # the askdirectory function will allow user to select directory to scan
        folder_path = filedialog.askdirectory()
        # Clear any existing text in the folder entry field
        folder_entry.delete(0, tk.END)
        # Insert the selected folder path into the entry field
        folder_entry.insert(0, folder_path)
```

```
# Define the "run_analysis" function
def run_analysis():
    # Use get() function to retrieve data from "question_entry" widget
    # Place data into "question" variable
    question = question_entry.get()
    # Use get() function to retrieve data from "folder_entry" widget
    # Place data into variable "folder"
    folder = folder_entry.get()
    # Run the "extract_breastfeeding_info" function
    # Use the folder and question variables in the function
    # Place the results of the function into the "results" variable
    results = extract_breastfeeding_info(folder, question)
    # Remove everything from the "results_text" widget
    results_text.delete('1.0', tk.END)
    # if there is data in the "results" list
    if results:
        # use lambda function to sort in reverse order
        # based on the 3rd element in each one
        results.sort(key=lambda x: x[2], reverse=True)
        # Using for loop, iterate through "doc", "answer" and "score" in
        results list
    for doc, answer, score in results:
        # insert variables "doc", "answer" and "score" in
        "results_text" widget
        results_text.insert(tk.END, f"Document: {doc}\nAnswer: {answer
        }\nConfidence: {score:.2f}\n\n")
else:
    results_text.insert(tk.END, "No answers found with sufficient
    confidence.")
        confidence.")
# Create the primary window with the "tk" toolkit using the Tk() class
# Assigned to variable "root" and called the root window
root = tk.Tk()
# Sets the title of the "root" window
# This can be changed
root.title("Breastfeeding information using local AI engine DistilBERT")
# Create a label widget in the main window called "Question" using pack()
method
tk.Label(root, text="Question:").pack()
# Create text Entry widget in main window, 50 characters wide
# Assign it to variable "question_entry"
question_entry = tk.Entry(root, width=50)
# Use pack() function to place in main window
question_entry.pack()
# Create a Label widget called "Folder"
# use pack() function to place in main window
tk.Label(root, text="Folder:").pack()
# Create text Entry widget with 50 character width
# Assign it to variable "folder_entry"
folder_entry = tk.Entry(root, width=50)
# Use pack() function to add it to the main window
folder_entry.pack()
```

```
# Create buttons below
# Create Button widget called "Select Folder"; use pack() function to
place in window
# When clicked, run "select_folder" command
tk.Button(root, text="Select Folder", command=select_folder).pack()
# Create Button widget called "Run Analysis"; use pack() function to
place in window
# When clicked, run "run_analysis" command
tk.Button(root, text="Run Analysis", command=run_analysis).pack()
# Create Text widget, 20x60 characters
# Associate with variable "results_text"
results_text = tk.Text(root, height=20, width=60)
# Use pack() function to put "results_text" into window
results_text.pack()
# Start Tkinter and listen for mouse clicks/movements, keyboard
# Keeps window open
root.mainloop()
```