Below is an example implementation that meets your requirements. This code:

* Uses **Tkinter** (with ttk widgets) to create a ChatGPT-like chat interface.
* Integrates with a local model via **LangChain** (here we assume an available wrapper such as an “Ollama” class—you may need to adjust initialization per your local setup).
* Saves chats (user input and bot responses) along with a timestamp.
* Attempts to save chat history directly to a Supabase table called "chats". When no internet connection is detected, the chats are cached locally in a JSON file and later synced when connectivity is restored.
* Provides a menu option to upload a PDF or Excel file so that you can later extract text, compute embeddings, and store them in Supabase (the file-processing function is a placeholder where you can integrate your own PDF/Excel parsing and embedding extraction logic).
* Uses threading so that model inference and syncing do not block the UI.
* Includes basic error logging (to a file named chatbot.log) and error messages via Tkinter’s message boxes.

Before running the code, please make sure that:

* You have installed the required packages (e.g., tkinter [built-in], langchain, supabase, and requests).
* You set up your Supabase credentials (either by setting the environment variables SUPABASE\_URL and SUPABASE\_KEY or by replacing the placeholders in the code).
* You adjust the model initialization (here using a hypothetical Ollama class from LangChain) to match your local model’s configuration.

Below is the complete code:

python

Copy

import os

import time

import json

import threading

import logging

import tkinter as tk

from tkinter import ttk, scrolledtext, filedialog, messagebox

import requests # Used to check internet connectivity

# --- Import LangChain and Supabase libraries ---

# NOTE: Adjust the following imports based on your local model and LangChain configuration.

try:

from langchain.llms import Ollama # Hypothetical integration with your local model

except ImportError:

# If the above is not available, create a dummy model class for testing.

class Ollama:

def \_\_init\_\_(self, model):

self.model = model

def \_\_call\_\_(self, prompt):

return f"Echo: {prompt}"

from supabase import create\_client, Client # Make sure you install supabase-py

# --- Configuration for Supabase ---

SUPABASE\_URL = os.environ.get("SUPABASE\_URL", "https://your-supabase-url.supabase.co")

SUPABASE\_KEY = os.environ.get("SUPABASE\_KEY", "your-supabase-api-key")

try:

supabase\_client: Client = create\_client(SUPABASE\_URL, SUPABASE\_KEY)

except Exception as e:

logging.error("Supabase client initialization error: %s", e)

supabase\_client = None

# --- Logging Configuration ---

logging.basicConfig(

level=logging.INFO,

filename='chatbot.log',

format='%(asctime)s - %(levelname)s - %(message)s'

)

# --- Utility Functions ---

def check\_internet\_connection() -> bool:

"""Simple check for internet connection by trying to reach a known URL."""

try:

requests.get("https://www.google.com", timeout=5)

return True

except Exception:

return False

def append\_to\_cache(chat\_entry: dict, cache\_file: str = "chat\_cache.json"):

"""Append a chat entry to the local cache (JSON file)."""

data = []

if os.path.exists(cache\_file):

try:

with open(cache\_file, "r") as f:

data = json.load(f)

except Exception as e:

logging.error("Failed to read cache file: %s", e)

data.append(chat\_entry)

try:

with open(cache\_file, "w") as f:

json.dump(data, f)

except Exception as e:

logging.error("Failed to write to cache file: %s", e)

def sync\_cached\_data(cache\_file: str = "chat\_cache.json"):

"""Attempt to sync cached chat entries to Supabase and clear cache on success."""

if not os.path.exists(cache\_file):

return

try:

with open(cache\_file, "r") as f:

data = json.load(f)

except Exception as e:

logging.error("Failed to read cache file: %s", e)

return

if not data:

return

if check\_internet\_connection() and supabase\_client is not None:

try:

for chat in data:

# Adjust table/column names as needed.

res = supabase\_client.table("chats").insert(chat).execute()

logging.info("Synced chat: %s", chat)

# Clear the cache after successful syncing.

os.remove(cache\_file)

except Exception as e:

logging.error("Error syncing to Supabase: %s", e)

else:

logging.info("Internet not available or Supabase client not initialized; sync deferred.")

# --- Main Application Class ---

class ChatbotApp(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Chatbot Desktop App")

self.geometry("600x700")

self.configure(bg="#f0f0f0")

# Initialize UI components.

self.create\_widgets()

# Initialize the local model via LangChain.

self.init\_local\_model()

# Start a background thread that periodically syncs cached data.

self.sync\_thread = threading.Thread(target=self.periodic\_sync, daemon=True)

self.sync\_thread.start()

def create\_widgets(self):

"""Create and layout the widgets."""

# --- Menu Bar ---

menubar = tk.Menu(self)

file\_menu = tk.Menu(menubar, tearoff=0)

file\_menu.add\_command(label="Upload PDF/Excel", command=self.upload\_file)

file\_menu.add\_separator()

file\_menu.add\_command(label="Exit", command=self.quit)

menubar.add\_cascade(label="File", menu=file\_menu)

self.config(menu=menubar)

# --- Chat Display Area ---

self.chat\_display = scrolledtext.ScrolledText(

self, wrap=tk.WORD, state='disabled', font=("Helvetica", 12)

)

self.chat\_display.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

# --- Input Frame ---

self.input\_frame = ttk.Frame(self)

self.input\_frame.pack(padx=10, pady=10, fill=tk.X)

# Text widget for multi-line user input.

self.user\_input = tk.Text(self.input\_frame, height=3, font=("Helvetica", 12))

self.user\_input.pack(side=tk.LEFT, fill=tk.X, expand=True, padx=(0, 10))

# Send button.

self.send\_button = ttk.Button(self.input\_frame, text="Send", command=self.on\_send)

self.send\_button.pack(side=tk.RIGHT)

def init\_local\_model(self):

"""Initialize the local language model using LangChain."""

try:

# Adjust the model name and parameters as required.

self.llm = Ollama(model="your\_local\_model\_name")

except Exception as e:

logging.error("Failed to initialize local model: %s", e)

messagebox.showerror("Error", "Failed to initialize local model.")

self.llm = None

def periodic\_sync(self):

"""Periodically check for an internet connection and sync cached data."""

while True:

if check\_internet\_connection():

sync\_cached\_data()

time.sleep(60) # Sync every 60 seconds

def on\_send(self):

"""Handle sending of a user message."""

user\_text = self.user\_input.get("1.0", tk.END).strip()

if not user\_text:

return

self.display\_message("User", user\_text)

self.user\_input.delete("1.0", tk.END)

# Process model response in a separate thread.

threading.Thread(target=self.get\_bot\_response, args=(user\_text,), daemon=True).start()

def display\_message(self, sender: str, message: str):

"""Display a message in the chat display area."""

self.chat\_display.configure(state='normal')

if sender == "User":

self.chat\_display.insert(tk.END, f"User: {message}\n", "user")

else:

self.chat\_display.insert(tk.END, f"Bot: {message}\n", "bot")

self.chat\_display.configure(state='disabled')

self.chat\_display.see(tk.END)

def get\_bot\_response(self, user\_text: str):

"""Query the local model for a response and handle saving."""

if self.llm is None:

bot\_response = "Local model not available."

else:

try:

# Query the local model via LangChain.

bot\_response = self.llm(user\_text)

except Exception as e:

logging.error("Error during model inference: %s", e)

bot\_response = "Error: Unable to get response from local model."

# Display the bot's response.

self.display\_message("Bot", bot\_response)

# Create a chat entry to be saved.

chat\_entry = {

"user\_message": user\_text,

"bot\_response": bot\_response,

"timestamp": time.time()

}

# Attempt to save to Supabase if online; otherwise, append to local cache.

if check\_internet\_connection() and supabase\_client is not None:

try:

res = supabase\_client.table("chats").insert(chat\_entry).execute()

logging.info("Chat saved to Supabase.")

except Exception as e:

logging.error("Failed to save chat to Supabase: %s", e)

append\_to\_cache(chat\_entry)

else:

append\_to\_cache(chat\_entry)

def upload\_file(self):

"""Open a file dialog to select a PDF or Excel file for processing."""

file\_path = filedialog.askopenfilename(

filetypes=[("PDF files", "\*.pdf"), ("Excel files", "\*.xlsx;\*.xls")]

)

if file\_path:

threading.Thread(target=self.process\_file, args=(file\_path,), daemon=True).start()

def process\_file(self, file\_path: str):

"""

Process the uploaded file (PDF/Excel) to extract text,

compute embeddings, and store them in Supabase.

(This is a placeholder function; integrate your parsing/embedding logic here.)

"""

try:

logging.info("Processing file: %s", file\_path)

# TODO: Implement text extraction from the file.

# For example:

# text = extract\_text\_from\_file(file\_path)

# from langchain.embeddings import OpenAIEmbeddings

# embeddings = OpenAIEmbeddings()

# embedding\_vector = embeddings.embed\_query(text)

# Save embedding\_vector to Supabase (e.g., in a "vectors" table).

# For now, just show an info message.

messagebox.showinfo("File Upload", "File processed and embeddings saved.")

except Exception as e:

logging.error("Error processing file: %s", e)

messagebox.showerror("File Upload Error", f"Error processing file: {e}")

# --- Run the Application ---

if \_\_name\_\_ == "\_\_main\_\_":

app = ChatbotApp()

app.mainloop()

**Next Steps**

1. **Adjust Model Integration:**  
   Replace the dummy or hypothetical Ollama integration with the actual code you use to communicate with your local model via LangChain.
2. **Set Up Supabase:**  
   Ensure your Supabase project has the tables (e.g., "chats" and (optionally) a "vectors" table) set up with appropriate columns, and adjust the insertion code if needed.
3. **Implement File Processing:**  
   In the process\_file method, implement the logic to extract text from PDFs/Excels, compute embeddings, and store them in your vector database.
4. **Testing Offline Mode:**  
   Test the application both with and without an internet connection to ensure the caching and syncing functions work as expected.

Feel free to adjust the styling (colors, fonts) or add additional features (like conversation history browsing, error logs view, etc.) to best suit your needs. Let me know if you have any further questions or need additional modifications!