```
import os
import json
import threading
import time
import base64
from datetime import datetime
from tkinter import *
from tkinter import filedialog, messagebox
from openai import OpenAI

# ------------- Configuración básica y seguridad -------------
# Usa SOLO las dos variables de entorno específicas.
API_KEYS = {
    "personal": os.getenv("OPENAI_API_KEY_PERSONAL", ""),
    "oposicion": os.getenv("OPENAI_API_KEY_OPOSICION", "")
}

def build_client():
    key = API_KEYS[current_api]
    if not key:
        raise RuntimeError(f"Falta la variable de entorno
OPENAI_API_KEY_{current_api.upper()}")
    return OpenAI(api_key=key)

# ------------- Carpetas de backups -------------
BASE_DIR = os.path.abspath(os.path.dirname(__file__))
BACKUP_DIRS = {
    "personal": os.path.join(BASE_DIR, "chat_backups_personal"),
    "oposicion": os.path.join(BASE_DIR, "chat_backups_oposicion")
}
for d in BACKUP_DIRS.values():
    os.makedirs(d, exist_ok=True)

# ------------- Variables globales de estado -------------
history = []  #
[{'role':'user'/'assistant','content':[{'type':'text','text':...}, ...]}]
current_api = "personal"
current_model = "gpt-5-codex"

# Modelos que requieren endpoint v1/responses (no chat/completions)
RESP_MODELS = {"gpt-5-codex", "gpt-5-complex"}

waiting_timer_id = None
waiting_start_ms = 0
waiting_label = None
waiting_frame = None

last_assistant_button = None
last_message_text_widget = None  # ahora guarda el último widget (Label) para
hacer scroll

client = build_client()

# ------------- Utilidades UI -------------
```

```
def is_at_bottom():
    try:
        lo, hi = canvas.yview()
        return hi >= 0.98
    except Exception:
        return True

def scroll_to_widget_end(widget):
    try:
        canvas.update_idletasks()
        chat_frame.update_idletasks()
        widget.update_idletasks()
        widget_bottom = widget.winfo_y() + widget.winfo_height()
        chat_h = max(1, chat_frame.winfo_height())
        canvas_h = max(1, canvas.winfo_height())
        if chat_h <= canvas_h:
            canvas.yview_moveto(1.0)
            return
        top_target = max(0, widget_bottom - canvas_h)
        max_top = max(1, chat_h - canvas_h)
        frac = 0.0 if max_top <= 0 else min(1.0, top_target / max_top)
        canvas.yview_moveto(frac)
    except Exception:
        pass

def start_waiting():
    global waiting_timer_id, waiting_start_ms, waiting_label, waiting_frame
    stop_waiting()
    waiting_start_ms = int(time.time() * 1000)
    wf = Frame(chat_frame, bg="#ffffff")
    wf.pack(fill="x", pady=(0, 5))
    lbl = Label(
        wf,
        text="☑ Enviado. Esperando respuesta… 00:00",
        bg="#E8F4FD",
        fg="#0b65c2",
        font=("Arial", 14),
        padx=14,
        pady=8,
        anchor="w",
        justify="left",
    )
    lbl.pack(anchor="w", padx=20, fill="x")

    def resize_wait_label(event=None, l=lbl, f=wf):
        try:
            l.config(wraplength=max(200, f.winfo_width() - 40))
        except Exception:
            pass
    resize_wait_label()
    wf.bind("<Configure>", resize_wait_label)

    if is_at_bottom():
        canvas.update_idletasks()
```

```
        canvas.yview_moveto(1.0)

    def tick():
        global waiting_timer_id
        if lbl.winfo_exists():
            elapsed = int(time.time() * 1000) - waiting_start_ms
            s = elapsed // 1000
            mm = str(s // 60).zfill(2)
            ss = str(s % 60).zfill(2)
            lbl.config(text=f"☑ Enviado. Esperando respuesta… {mm}:{ss}")
            waiting_timer_id = root.after(500, tick)

    waiting_frame = wf
    waiting_label = lbl
    tick()

def stop_waiting(final_text=None):
    global waiting_timer_id, waiting_label, waiting_frame
    if waiting_timer_id:
        try:
            root.after_cancel(waiting_timer_id)
        except Exception:
            pass
        waiting_timer_id = None
    if waiting_label and final_text:
        waiting_label.config(text=final_text)
    if waiting_frame:
        wf = waiting_frame
        waiting_label = None
        waiting_frame = None
        root.after(1500, lambda: wf.destroy())

def finish_waiting(ok=True):
    ms = int(time.time() * 1000) - waiting_start_ms
    s = ms // 1000
    if ok:
        stop_waiting(f"☑ Respuesta recibida en {s}s")
    else:
        stop_waiting(f"⚠ Error. Tiempo transcurrido: {s}s")

def autosize_text_widget(w: Text):
    try:
        was_disabled = (str(w.cget("state")) == "disabled")
        if was_disabled:
            w.configure(state="normal")
        w.update_idletasks()
        count = w.count("1.0", "end-1c", "displaylines")
        lines = int(count[0]) if count else 1
        w.configure(height=max(1, lines))
        if was_disabled:
            w.configure(state="disabled")
    except Exception:
        pass
```

```
def reflow_text_heights():
    pass

def copy_to_clipboard(text, btn=None):
    try:
        root.clipboard_clear()
        root.clipboard_append(text)
        if btn:
            btn.config(text="Copiado ✓")
            root.after(1200, lambda: btn.config(text="Copiar"))
    except Exception as e:
        messagebox.showerror("Copiar", f"No se pudo copiar: {e}")

def set_last_message_text_widget(w):
    global last_message_text_widget
    last_message_text_widget = w

def _on_request_done(answer, ok):
    finish_waiting(ok)
    add_message("assistant", answer)
    send_button.config(state="normal")

def ask_large_string(title, prompt, initialvalue="", width_px=700,
height_px=300):
    dlg = Toplevel(root)
    dlg.title(title)
    dlg.transient(root)
    dlg.configure(bg="#ffffff")
    dlg.grab_set()
    try:
        root.update_idletasks()
        rx, ry = root.winfo_rootx(), root.winfo_rooty()
        rw, rh = root.winfo_width(), root.winfo_height()
        x = rx + max(0, (rw - width_px) // 2)
        y = ry + max(0, (rh - height_px) // 2)
        dlg.geometry(f"{width_px}x{height_px}+{x}+{y}")
    except Exception:
        dlg.geometry(f"{width_px}x{height_px}")

    Label(dlg, text=prompt, bg="#ffffff", font=("Arial", 13)).pack(anchor="w",
padx=14, pady=(12, 6))
    txt = Text(dlg, wrap="word", height=8, font=("Arial", 13), bg="#f7f7f8")
    txt.pack(fill="both", expand=True, padx=14)
    if initialvalue:
        txt.insert("1.0", initialvalue)

    btn_frame = Frame(dlg, bg="#ffffff")
    btn_frame.pack(fill="x", padx=14, pady=10)

    result = {"val": None}
    def on_ok():
        result["val"] = txt.get("1.0", "end-1c").strip()
        dlg.destroy()
```

```
    def on_cancel():
        result["val"] = None
        dlg.destroy()

    Button(btn_frame, text="Cancelar", width=12,
command=on_cancel).pack(side=RIGHT, padx=5)
    Button(btn_frame, text="Aceptar", width=12, command=on_ok, bg="#10a37f",
fg="white").pack(side=RIGHT, padx=5)

    dlg.bind("<Return>", lambda e: on_ok())
    dlg.bind("<Escape>", lambda e: on_cancel())
    txt.focus_set()
    dlg.wait_window()
    return result["val"]

# ------------- Chat básico (texto) con chat/completions o responses
-------------
def build_chat_messages_for_api():
    messages = []
    for msg in history:
        if msg.get("role") not in ("user", "assistant"):
            continue
        text_parts = [c.get("text", "") for c in msg.get("content", []) if
c.get("type") == "text"]
        if not text_parts:
            continue
        content = "\n".join([p for p in text_parts if p])
        if not content.strip():
            continue
        messages.append({"role": msg["role"], "content": content})
    return messages

def ask_text(model, messages):
    if model in RESP_MODELS:
        prompt = "\n\n".join(f"{m['role']}: {m['content']}" for m in messages)
        resp = client.responses.create(model=model, input=prompt)
        return _extract_response_text(resp)
    else:
        resp = client.chat.completions.create(model=model, messages=messages)
        return resp.choices[0].message.content

def send_message():
    user_input = input_text.get("1.0", END).strip()
    if not user_input:
        return
    input_text.delete("1.0", END)
    add_message("user", user_input)
    history.append({"role": "user", "content": [{"type": "text", "text":
user_input}]})
    start_waiting()
    send_button.config(state="disabled")
    threading.Thread(target=generate_response, daemon=True).start()

def generate_response():
```

```
    ok = True
    answer = ""
    try:
        messages = build_chat_messages_for_api()
        answer = ask_text(current_model, messages)
    except Exception as e:
        ok = False
        answer = f"⚠ Error: {e}"
    history.append({"role": "assistant", "content": [{"type": "text", "text":
answer}]})
    root.after(0, lambda ans=answer, ok=ok: _on_request_done(ans, ok))

def add_message(role, content_text):
    global last_assistant_button
    bottom_before = is_at_bottom()
    bg_color, anchor_side = ("#DCF8C6", "e") if role == "user" else ("#F1F0F0",
"w")
    text_color = "black"

    row = Frame(chat_frame, bg="#ffffff")
    row.pack(fill="x", pady=5)

    holder = Frame(row, bg="#ffffff")
    holder.pack(anchor=anchor_side, padx=20, fill="x")

    lbl = Label(
        holder,
        text=content_text,
        bg=bg_color,
        fg=text_color,
        justify="left",
        font=("Arial", 18),
        anchor="w",
        padx=20,
        pady=10
    )
    lbl.pack(fill="x")

    def resize_label(event=None, l=lbl, h=holder):
        try:
            l.config(wraplength=max(200, h.winfo_width() - 40))
        except Exception:
            pass
    resize_label()
    holder.bind("<Configure>", resize_label)

    if role == "assistant":
        if last_assistant_button and last_assistant_button.winfo_exists():
            try:
                last_assistant_button.destroy()
            except Exception:
                pass
        copy_btn = Button(
            row, text="Copiar", font=("Arial", 12),
```

```
            padx=10, pady=3, bg="#E8F4FD", fg="#0b65c2",
            relief="groove", cursor="hand2"
        )
        copy_btn.config(command=lambda: copy_to_clipboard(content_text,
copy_btn))
        copy_btn.pack(anchor="e", padx=24, pady=(2, 2))
        last_assistant_button = copy_btn

    def finalize():
        if bottom_before:
            scroll_to_widget_end(lbl)

    root.after_idle(lambda: (finalize(), set_last_message_text_widget(lbl)))

# ------------- Helper para Responses -------------
def _extract_response_text(resp):
    try:
        return resp.output_text
    except Exception:
        try:
            parts = []
            for item in getattr(resp, "output", []) or []:
                for c in getattr(item, "content", []) or []:
                    t = getattr(c, "type", "")
                    if t in ("output_text", "text"):
                        txt_obj = getattr(c, "text", None)
                        if txt_obj is not None:
                            val = getattr(txt_obj, "value", None)
                            if val:
                                parts.append(val)
                        elif hasattr(c, "text"):
                            parts.append(c.text)
            return "\n".join([p for p in parts if p]) or "(sin texto)"
        except Exception:
            return "(sin texto)"

# ------------- Análisis de archivo con Responses (sin attachments)
-------------
def ask_about_file_assistants(file_path, question, model=None):
    """
    Sube el archivo (purpose='assistants') y pregunta usando la Responses API.
    - Imágenes: input_image (con data URL).
    - Documentos: input_file (con file_id) — sin usar 'attachments' ni
'file_search'.
    Devuelve (file_id, respuesta).
    """
    mdl = model or current_model

    with open(file_path, "rb") as f:
        up = client.files.create(file=f, purpose="assistants")
    file_id = up.id

    name = os.path.basename(file_path).lower()
    is_image = name.endswith((".png", ".jpg", ".jpeg", ".gif", ".webp"))
```

```
    try:
        if is_image:
            data_url = image_to_data_url(file_path)
            resp = client.responses.create(
                model=mdl,
                input=[
                    {
                        "role": "user",
                        "content": [
                            {"type": "input_text", "text": question},
                            {"type": "input_image", "image_url": data_url},
                        ],
                    }
                ],
            )
            return file_id, _extract_response_text(resp)
        else:
            # Enviamos el archivo por su file_id como 'input_file'
            resp = client.responses.create(
                model=mdl,
                input=[
                    {
                        "role": "user",
                        "content": [
                            {"type": "input_text", "text": question},
                            {"type": "input_file", "file_id": file_id},
                        ],
                    }
                ],
            )
            return file_id, _extract_response_text(resp)
    except Exception:
        raise

def action_analyze_file_assistants():
    file_path = filedialog.askopenfilename(
        filetypes=[("Documentos e Imágenes",
"*.pdf;*.txt;*.md;*.py;*.json;*.csv;*.png;*.jpg;*.jpeg;*.gif;*.webp")]
    )
    if not file_path:
        return
    q = ask_large_string("Pregunta", "¿Qué quieres preguntar sobre el archivo?",
initialvalue="Resume el archivo, por favor.")
    if not q:
        return

    filename = os.path.basename(file_path)
    add_message("user", f"Analizando archivo con Assistants:
{filename}\nPregunta: {q}")
    history.append({"role": "user", "content": [{"type": "text", "text":
f"Analizando archivo con Assistants: {filename}\nPregunta: {q}"}]})
    start_waiting()
    send_button.config(state="disabled")
```

```
    def worker():
        ok = True
        ans = ""
        try:
            file_id, ans = ask_about_file_assistants(file_path, q,
model=current_model)
            ans = f"(file_id: {file_id})\n\n{ans}"
        except Exception as e:
            ok = False
            ans = f"⚠ Error analizando archivo: {e}"
        history.append({"role": "assistant", "content": [{"type": "text",
"text": ans}]})
        root.after(0, lambda: _on_request_done(ans, ok))

    threading.Thread(target=worker, daemon=True).start()

# ------------- Análisis de imagen con base64 (ramificación por modelo)
-------------
def image_to_data_url(path):
    name = os.path.basename(path).lower()
    if name.endswith(".png"):
        mime = "image/png"
    elif name.endswith(".jpg") or name.endswith(".jpeg"):
        mime = "image/jpeg"
    elif name.endswith(".gif"):
        mime = "image/gif"
    elif name.endswith(".webp"):
        mime = "image/webp"
    else:
        mime = "image/jpeg"
    with open(path, "rb") as f:
        b64 = base64.b64encode(f.read()).decode("utf-8")
    return f"data:{mime};base64,{b64}"

def analyze_image_base64(file_path, question, model=None):
    mdl = model or current_model
    data_url = image_to_data_url(file_path)
    if mdl in RESP_MODELS:
        resp = client.responses.create(
            model=mdl,
            input=[{
                "role": "user",
                "content": [
                    {"type": "input_text", "text": question},
                    {"type": "input_image", "image_url": data_url}
                ]
            }]
        )
        return _extract_response_text(resp)
    else:
        response = client.chat.completions.create(
            model=mdl,
            messages=[{
```

```
                "role": "user",
                "content": [
                    {"type": "text", "text": question},
                    {"type": "image_url", "image_url": {"url": data_url}}
                ]
            }]
        )
        return response.choices[0].message.content

def action_analyze_image_base64():
    file_path = filedialog.askopenfilename(
        filetypes=[("Imágenes", "*.png;*.jpg;*.jpeg;*.gif;*.webp")]
    )
    if not file_path:
        return
    q = ask_large_string("Pregunta", "¿Qué quieres preguntar sobre la imagen?",
initialvalue="Describe el contenido de la imagen.")
    if not q:
        return

    filename = os.path.basename(file_path)
    add_message("user", f"Analizando imagen por base64: {filename}\nPregunta:
{q}")
    history.append({"role": "user", "content": [{"type": "text", "text":
f"Analizando imagen por base64: {filename}\nPregunta: {q}"}]})
    start_waiting()
    send_button.config(state="disabled")

    def worker():
        ok = True
        ans = ""
        try:
            ans = analyze_image_base64(file_path, q, model=current_model)
        except Exception as e:
            ok = False
            ans = f"⚠ Error analizando imagen: {e}"
        history.append({"role": "assistant", "content": [{"type": "text",
"text": ans}]})
        root.after(0, lambda: _on_request_done(ans, ok))

    threading.Thread(target=worker, daemon=True).start()

# ------------- Varias (exportar, importar, etc.) -------------
def change_model(selected):
    global current_model
    current_model = selected

def change_api(selected):
    global current_api, client
    current_api = selected
    client = build_client()

def export_chat():
    if not history:
```

```
        messagebox.showinfo("Exportar","No hay mensajes para exportar")
        return
    filename =
f"{current_api}_export_{datetime.now().strftime('%Y-%m-%dT%H-%M-%S')}.json"
    path = os.path.join(BACKUP_DIRS[current_api], filename)
    with open(path, "w", encoding="utf-8") as f:
        json.dump(history, f, indent=2, ensure_ascii=False)
    messagebox.showinfo("Exportar", f"Chat exportado en:\n{path}")

def import_chat():
    file_path = filedialog.askopenfilename(filetypes=[("JSON Files","*.json")])
    if not file_path:
        return
    try:
        with open(file_path, "r", encoding="utf-8") as f:
            data = json.load(f)
        if isinstance(data, list):
            global history
            history = data
            refresh_chat()
            messagebox.showinfo("Importar","Chat importado correctamente")
        else:
            messagebox.showerror("Error","Formato incorrecto")
    except Exception as e:
        messagebox.showerror("Error", f"No se pudo importar: {e}")

def clear_chat():
    if messagebox.askyesno("Borrar", "¿Deseas borrar todo el chat actual?"):
        save_auto_chat("before_clear")
        global history
        history = []
        refresh_chat()

def refresh_chat():
    for w in chat_frame.winfo_children():
        w.destroy()
    global last_assistant_button, last_message_text_widget
    last_assistant_button = None
    last_message_text_widget = None
    for m in history:
        for c in m.get("content", []):
            if c.get("type") == "text":
                add_message(m["role"], c.get("text", ""))
            else:
                add_message(m["role"], f"[{c.get('type','obj')}]")

def save_auto_chat(prefix="chat_auto"):
    if not history:
        return
    filename =
f"{current_api}_{prefix}_{datetime.now().strftime('%Y-%m-%dT%H-%M-%S')}.json"
    path = os.path.join(BACKUP_DIRS[current_api], filename)
    try:
        with open(path, "w", encoding="utf-8") as f:
```

```
            json.dump(history, f, indent=2, ensure_ascii=False)
    except Exception as e:
        print("⚠ Error guardando chat automático:", e)

def copy_all_chat():
    text = ""
    for msg in history:
        for c in msg.get("content", []):
            if c.get("type") == "text":
                prefix = "🧑 Tú: " if msg["role"] == "user" else "🤖 Asistente:
"
                text += f"{prefix}{c['text']}\n\n"
    root.clipboard_clear()
    root.clipboard_append(text)
    messagebox.showinfo("Copiar todo", "Chat copiado al portapapeles.")

# ------------- GUI -------------
root = Tk()
root.title("💬 Chat con archivos (Assistants y Base64)")
sw, sh = root.winfo_screenwidth(), root.winfo_screenheight()
root.geometry(f"{int(sw*0.9)}x{int(sh*0.9)}+{int(sw*0.05)}+{int(sh*0.05)}")
root.configure(bg="#ffffff")

top_frame = Frame(root, bg="#ffffff")
top_frame.pack(fill=X, padx=15, pady=6)

top_label_font = ("Arial", 12)
top_button_font = ("Arial", 12, "bold")

Label(top_frame, text="Modelo:", bg="#ffffff",
font=top_label_font).pack(side=LEFT)

model_var = StringVar(value=current_model)
model_menu = OptionMenu(
    top_frame, model_var,
    "gpt-5", "gpt-5-codex", "gpt-5-complex", "gpt-5-chat-latest", "gpt-5-mini",
"gpt-5-nano",
    command=change_model
)
model_menu.config(font=top_label_font, width=14)
model_menu["menu"].config(font=top_label_font)
model_menu.pack(side=LEFT, padx=8)

Label(top_frame, text="Clave API:", bg="#ffffff",
font=top_label_font).pack(side=LEFT, padx=(16, 0))
api_var = StringVar(value=current_api)
api_menu = OptionMenu(top_frame, api_var, "personal", "oposicion",
command=change_api)
api_menu.config(font=top_label_font, width=12)
api_menu["menu"].config(font=top_label_font)
api_menu.pack(side=LEFT, padx=8)

Button(top_frame, text="Analizar archivo", font=top_button_font,
command=action_analyze_file_assistants, width=20, height=1).pack(side=RIGHT,
```

```
padx=4)
Button(top_frame, text="Analizar imagen", font=top_button_font,
command=action_analyze_image_base64, width=18, height=1).pack(side=RIGHT,
padx=4)
Button(top_frame, text="Copiar todo", font=top_button_font,
command=copy_all_chat, width=10, height=1).pack(side=RIGHT, padx=4)
Button(top_frame, text="Exportar", font=top_button_font, command=export_chat,
width=9, height=1).pack(side=RIGHT, padx=4)
Button(top_frame, text="Importar", font=top_button_font, command=import_chat,
width=9, height=1).pack(side=RIGHT, padx=4)
Button(top_frame, text="Borrar", font=top_button_font, command=clear_chat,
width=8, height=1).pack(side=RIGHT, padx=4)

input_frame = Frame(root, bg="#ffffff", bd=2, relief="groove")
input_frame.pack(side=BOTTOM, fill=X, padx=15, pady=5)
input_scroll = Scrollbar(input_frame)
input_scroll.pack(side=RIGHT, fill=Y)
input_text = Text(input_frame, height=6, font=("Arial", 18), bg="#f7f7f8",
wrap="word", yscrollcommand=input_scroll.set)
input_text.pack(side=LEFT, fill=X, expand=True, padx=(0, 15))
input_scroll.config(command=input_text.yview)

send_button_font = ("Arial", 16, "bold")
send_button = Button(input_frame, text="Enviar", font=send_button_font,
width=14, height=2, bg="#10a37f", fg="white", command=send_message)
send_button.pack(side=RIGHT)

chat_canvas_frame = Frame(root)
chat_canvas_frame.pack(fill=BOTH, expand=True, padx=15, pady=(0, 5))
canvas = Canvas(chat_canvas_frame, bg="#ffffff", highlightthickness=0)
canvas.pack(side=LEFT, fill=BOTH, expand=True)

def on_scrollbar(*args):
    canvas.yview(*args)

scrollbar = Scrollbar(chat_canvas_frame, orient=VERTICAL, command=on_scrollbar)
scrollbar.pack(side=RIGHT, fill=Y)
canvas.configure(yscrollcommand=scrollbar.set)

chat_frame = Frame(canvas, bg="#ffffff")
chat_window = canvas.create_window((0, 0), window=chat_frame, anchor="nw")

def on_chat_frame_configure(event):
    canvas.configure(scrollregion=canvas.bbox("all"))

def on_canvas_configure(event):
    canvas.itemconfig(chat_window, width=event.width)

chat_frame.bind("<Configure>", on_chat_frame_configure)
canvas.bind("<Configure>", on_canvas_configure)

def _on_mousewheel(event):
    try:
        if event.num == 4:
```

```
            canvas.yview_scroll(-3, "units")
        elif event.num == 5:
            canvas.yview_scroll(3, "units")
        else:
            delta = int(-1 * (event.delta / 120)) if hasattr(event, "delta")
else -1
            canvas.yview_scroll(delta * 3, "units")
        return "break"
    except Exception:
        pass

canvas.bind("<MouseWheel>", _on_mousewheel)
canvas.bind("<Button-4>", _on_mousewheel)
canvas.bind("<Button-5>", _on_mousewheel)
chat_frame.bind("<MouseWheel>", _on_mousewheel)
chat_frame.bind("<Button-4>", _on_mousewheel)
chat_frame.bind("<Button-5>", _on_mousewheel)

def on_closing():
    save_auto_chat()
    try:
        if waiting_timer_id:
            root.after_cancel(waiting_timer_id)
    except Exception:
        pass
    root.destroy()

root.protocol("WM_DELETE_WINDOW", on_closing)
root.mainloop()
```