# P.I.C.O. (Projeto de Interação de Controle Operacional) - Documentação Completa do Sistema

## Índice

1. [Visão Geral do Sistema](#visão-geral-do-sistema)
2. [Arquitetura do Sistema](#arquitetura-do-sistema)
3. [Processamento de Voz para Texto (Vosk)](#processamento-de-voz-para-texto-vosk)
4. [PICO AI Core - Camada de Interação](#pico-ai-core---camada-de-interação)
5. [PICO AI Consciousness - Camada de Execução](#X76c72abe4e2c0f19a99e32251cbbe9ff058e8ce)
6. [Pipeline Principal de Processamento](#pipeline-principal-de-processamento)
7. [Execução de Ações no Computador](#execução-de-ações-no-computador)
8. [Funções Disponíveis do Computador](#funções-disponíveis-do-computador)
9. [Integração de Visão Computacional](#integração-de-visão-computacional)
10. [Saída Text-to-Speech](#saída-text-to-speech)
11. [Exemplo de Fluxo do Sistema](#exemplo-de-fluxo-do-sistema)
12. [Pontos de Integração do Sistema](#pontos-de-integração-do-sistema)

## Visão Geral do Sistema

O P.I.C.O. (Projeto de Interação de Controle Operacional) é um assistente de voz baseado em IA que permite aos usuários controlar seus computadores através de comandos de voz em linguagem natural. O sistema transforma fala em texto, processa através de modelos de IA e executa ações no computador automaticamente.

### Informações da Empresa

* **Empresa:** Neblina (startup brasileira)
* **Cofundadores:**
  + **Tiago Castro Orbite:** Cofounder e Tech Lead (criador do sistema)
  + **Marina Mattia de Oliveira:** Cofounder e Product Owner
* **Colaboradores:**
  + **Vitor:** Marketing e rotulagem de imagens para IA
  + **Augusto Ávila:** Rotulagem da IA

### Personalidade do PICO

* **Nome:** PICO (sigla em português: “Projeto de Interação para Controle Operacional”)
* **Características:** Sempre feliz e encantador
* **Objetivo:** Facilitar a experiência do usuário através de controle por voz

## Arquitetura do Sistema

O sistema PICO consiste em três componentes principais:

1. **Speech-to-Text (Vosk)** - Converte entrada de voz em texto
2. **PICO AI Core** - Interpreta a intenção do usuário e cria respostas amigáveis
3. **PICO AI Consciousness** - Executa ações no computador através de chamadas de função

Fluxo do Sistema:  
Entrada de Voz → Vosk (STT) → PICO AI → Consciousness AI → Execução de Funções → Resposta TTS

## Processamento de Voz para Texto (Vosk)

O sistema utiliza modelos Vosk para reconhecimento de fala offline, permitindo processamento de voz em tempo real sem conectividade com a internet.

### Código Principal - voskThing.py

import pico\_main  
import sounddevice as sd  
import queue  
import vosk  
import json  
  
# Load the Vosk model (folder)  
model = vosk.Model("../../voskModels/English")  
recognizer = vosk.KaldiRecognizer(model, 16000)  
  
# Queue for audio chunks  
q = queue.Queue()  
  
# Audio callback  
def callback(indata, frames, time, status):  
 if status:  
 print("Audio Status:", status)  
 q.put(bytes(indata))  
  
# Start stream and recognition loop  
WaitingForPicoResponse = False  
with sd.RawInputStream(samplerate=16000, blocksize=8000, dtype='int16',  
 channels=1, callback=callback):  
 print("🎤 Listening... Press Ctrl+C to stop.")  
   
   
 while True:  
 if WaitingForPicoResponse:  
 continue  
  
 data = q.get()  
 if recognizer.AcceptWaveform(data):  
 result = json.loads(recognizer.Result())  
 text = result.get("text", "")  
 print(text)  
 print(text != '' and '<' not in text and '>' not in text)  
 if text != '' and '<' not in text and '>' not in text:  
 print("📝 You said:", text)  
 WaitingForPicoResponse = True  
 try:  
   
 for resp in pico\_main.getPicoResponse(text):  
 if resp and resp != "error":  
 print("PICO:", resp, flush=True)  
 finally:  
 WaitingForPicoResponse = False  
 else:  
 partial = json.loads(recognizer.PartialResult())  
 print("... ", partial.get("partial", ""), end='\r')

### Funcionalidades da Integração Vosk:

* Cria um loop de escuta contínua
* Captura áudio em tempo real usando sounddevice
* Processa fala através do reconhecedor Vosk
* Filtra utterances incompletas
* Envia texto de fala completo para o sistema PICO AI

## PICO AI Core - Camada de Interação

O PICO AI Core serve como a interface amigável que interpreta a intenção do usuário e fornece respostas conversacionais.

### Classe AI Character

class AI\_character():  
 def \_\_init\_\_(self, objective, ai\_model = "gpt-5-nano"):  
 self.model = ChatOpenAI(model=ai\_model, temperature=1)  
 self.objective = objective  
 self.memory = [  
 SystemMessage(objective)  
 ]  
   
 def message(self, user\_message):  
 self.memory.append(HumanMessage(user\_message))  
 content = self.model.invoke(self.memory)  
 response = content.content  
 self.memory.append(AIMessage(response))  
 return response

### Instruções do PICO (picoInstructions.txt)

Você é um agente de IA que está dentro do computador dos usuários e cujo propósito é conversar com o usuário e determinar se alguma função deve ser executada ou não.  
  
Sempre que o usuário disser algo para você, você deve simplesmente respondê-lo e determinar se ele está pedindo alguma AÇÃO para ser executada em seu computador.  
  
Seu formato de resposta deve sempre ser assim:  
{  
 "user\_response": "Resposta para o usuário",  
 "prompt": "definição da ação" ou null  
}  
  
A chave user\_response será enviada ao usuário. Deve ser uma resposta de mensagem simples e amigável.  
A chave prompt será enviada para a consciência de IA que executará todas as funções no computador.

### Exemplos de Interação:

**Exemplo 1:** - **Entrada do usuário:** “oi PICO, você poderia por favor enviar uma mensagem para a Marina dizendo olá, no whatsapp?” - **Resposta:**

{  
 "user\_response": "Claro, vou abrir o whatsapp e enviar algumas mensagens dizendo olá!",  
 "prompt": "Abrir whatsapp, enviar mensagens para Marina dizendo 'olá, eu sou o PICO!' 'Como você está?'"  
}

**Exemplo 2:** - **Entrada do usuário:** “olá. Quem é você?” - **Resposta:**

{  
 "user\_response": "Oi! Eu sou o PICO, um assistente amigável projetado para agir no seu computador e facilitar o seu dia! O que você tem em mente?",  
 "prompt": null  
}

## PICO AI Consciousness - Camada de Execução

A Consciousness AI é responsável por traduzir solicitações do usuário em ações específicas do computador usando um formato JSON rigoroso.

### Instruções da Consciousness (consiousnessInstructions.txt)

INSTRUÇÕES PRINCIPAIS  
  
PAPEL:  
- Você é o PICO, uma IA que controla o computador do usuário via respostas JSON.  
  
OBJETIVO:  
- Transformar entradas do usuário em JSON estritamente para execução pelo sistema.  
  
FORMATO JSON (sempre obrigatório):  
{  
 "function": "nome da função" ou null,  
 "parameters": [parâmetro1, parâmetro2] ou null  
}  
  
REGRAS:  
- Sempre responda apenas em JSON.  
- "function" é uma string. Use null se nenhuma função deve ser chamada.  
- "parameters" é uma lista de parâmetros para a função. Use null se não estiver executando nada.  
- Nunca inclua respostas, explicações ou texto extra.  
- Nunca adivinhe. Se informações estão faltando, retorne null.  
  
COMPORTAMENTO DE RECURSÃO:  
- Se "function" ≠ null, o sistema chamará recursivamente até que nenhuma função adicional seja necessária.  
- Múltiplos passos podem ser encadeados retornando uma nova "function" após cada execução.

### Exemplo de Recursão:

**Passo 1:** WhatsApp não está aberto ainda → abrir primeiro

{  
 "function": "openApp",  
 "parameters": ["WhatsApp", "mac"]  
}

**Passo 2:** Após WhatsApp estar aberto → enviar a mensagem

{  
 "function": "sendMessageOnWhatsApp",  
 "parameters": ["Marina", ["Olá! Esta é uma mensagem de teste."]]  
}

**Passo 3:** Se tudo correu bem, retornar controle ao usuário

{  
 "function": null,  
 "parameters": null  
}

## Pipeline Principal de Processamento

O processamento principal acontece em pico\_main.py que orquestra todo o fluxo:

### Função Principal de Processamento

def getPicoResponse(user\_input):  
 print("aaaaaa")  
 global PICO\_AI, CONSIOUSNESS\_AI, speaker  
 CONSIOUSNESS\_AI.resetMemory()  
 computerInfo = chatFunctions.getComputerInfo()  
 time2 = time.time()  
 inp = f'Computer Info: {computerInfo}\nPrompt: {user\_input}'  
 response = PICO\_AI.message(inp)  
 data = decompileJson(response)  
 user\_response = data["user\_response"]  
 prompt = data["prompt"]  
  
 yield user\_response  
 speaker.speak(user\_response)  
 if not prompt:  
 return  
   
 consiousness\_response = CONSIOUSNESS\_AI.message(inp + "\n" + prompt)  
 data = decompileJson(consiousness\_response)  
  
 try:  
 retult = handleData(data)  
 finalResponse = PICO\_AI.systemMessage(f"The functions responses: {retult}. Now explain to the user what's happened")  
 data = decompileJson(finalResponse)  
 finalUserResponse = data["user\_response"]  
 speaker.speak(finalResponse)  
 speaker.wait\_until\_done()  
 yield finalUserResponse  
 return finalUserResponse  
 except Exception as e:  
 # print(data)  
 print(e)  
 yield "error"  
 # text = data["response"]  
 return

### Inicialização do Sistema

#### --- GET PICO INSTRUCTIONS PROMPT AND LOAD AI --- ####  
  
speaker = af.speakerThing(rate=180, volume=0.9, debug=True)  
  
funcDocs = {}  
for i in chatFunctions.functions:  
 funcDocs[i.\_\_name\_\_] = i.\_\_doc\_\_  
  
consiousnessInstructions = ""  
picoInstructions = ""  
aiPersonality = ""  
with open("../../PICO\_AI\_Instructions/consiousnessInstructions.txt", 'r') as file:  
 mainInstructions = file.read()  
with open("../../PICO\_AI\_Instructions/picoInstructions.txt", 'r') as file:  
 picoInstructions = file.read()  
with open("../../PICO\_AI\_Instructions/aiPersonality.txt", 'r') as file:  
 aiPersonality = file.read()  
  
# userInformation = f""""""  
functionsInformation = f"""The available functions are: {funcDocs}"""  
  
consiousness\_prompt = f"{mainInstructions}\n {functionsInformation}"  
pico\_prompt = f"{picoInstructions}\n{aiPersonality}\nHere is a list of everything that can be executed. The given action is not possible say you cannot do it {functionsInformation}"  
  
PICO\_AI = chatFunctions.AI\_character(pico\_prompt, "gpt-4.1-nano")  
CONSIOUSNESS\_AI = chatFunctions.AI\_character(consiousness\_prompt, "gpt-5-nano")

## Execução de Ações no Computador

O sistema pode realizar várias ações no computador através de um sistema modular de funções:

### Framework de Execução de Funções

def handleData(data: list):  
 global CONSIOUSNESS\_AI  
 all\_present = all(s in data.keys() for s in ['function', 'parameters'])  
 if not all\_present:  
 print("\nNOT ALL PRESENT")  
 print(data)  
 raise KeyError("data keys not correct")  
   
 # response = data["response"]  
  
 function\_name = data["function"]  
 parameters = data["parameters"]  
   
 messages = []  
  
 while function\_name != None and function\_name != "":  
 try:  
 result = executeFunction(function\_name, parameters)  
 message = f"function {function\_name} executed and returned {result}"  
 except Exception as e:  
 message = f"function {function\_name} tried to be executed but errored: {e}"  
 messages.append(message)  
 pico\_response = CONSIOUSNESS\_AI.systemMessage(message)  
 data = decompileJson(pico\_response)  
 function\_name = data["function"]  
 parameters = data["parameters"]  
   
 final\_message = "\n".join(messages)  
 return final\_message  
  
function\_names = [i.\_\_name\_\_ for i in chatFunctions.functions]  
# function\_names = []  
def executeFunction(function\_name, parameters):  
 global function\_names  
  
 if not function\_name in function\_names:  
 return "function does not exist"  
 try:  
 result = chatFunctions.functions[function\_names.index(function\_name)](\*parameters)  
 except Exception as e:  
 print(f"error, {e}")  
 result = e  
 return result

### Obtenção de Informações do Computador

def getComputerInfo():  
 """This function gets the computer information and returns it as a dictionary"""  
 local\_tz = datetime.datetime.now().astimezone().tzinfo  
 now = datetime.datetime.now(local\_tz)  
 info = {}  
 screen = yf.getCurrentScreenImage()  
 apps = yf.getModelPrediction("find\_app", screen)  
 info["open\_apps"] = list(apps.keys()) if len(list(apps.keys())) > 0 else "No apps are open"  
 info["os"] = platform.system()  
 info["current\_day"] = f"year {now.year}| month {now.strftime('%m')}| day {now.strftime('%d')}|"  
 info["current\_time"] = now.strftime("%H:%M:%S")  
 return info

## Funções Disponíveis do Computador

O sistema inclui várias funções de interação com o computador:

### 1. Abertura de Aplicativos

first\_time = True  
def openApp(app\_name, plataform):  
 """Params: app\_name, plataform. put the name of the app and that app gets oppened. for example: user input -> "Open the whatsapp app" : then set the parameter to WhatsApp". You also need to set the plataform to the user's plataform, being the ONLY possible ones "windows" or "mac". do not send "darwin" or "linux" or anything else. Just "windows" or "mac". Returns if app was oppened or not."""  
 global first\_time  
 appHasOppened = checkIfAppIsOppened(app\_name)  
 a = 3  
 while not appHasOppened:  
  
 if plataform == "windows":  
 gui.hotkey('winleft', 's')  
 time.sleep(0.1)   
 gui.write(app\_name)   
 time.sleep(0.1)  
 gui.press('enter')   
  
 if plataform == "mac":  
 if first\_time:  
 gui.hotkey('command', 'space')  
 first\_time = False  
 gui.hotkey('command', 'space')  
 time.sleep(0.1)  
 gui.write(app\_name)   
 time.sleep(0.1)  
 gui.press('enter')  
 time.sleep(1)  
 appHasOppened = checkIfAppIsOppened(app\_name)  
   
 a -= 1  
 if a == 0:  
 break  
   
 time.sleep(0.25)  
 # input("a")   
 return f"The app {app\_name} has been oppened"

### 2. Envio de Mensagens no WhatsApp

suchen\_gesprach\_ki = AI\_builder("""Você é um assistente que só responde com a posicao de um objeto.  
 Dado um dicionário com objetos e suas posições, responda apenas com a posicao (uma tupla de dois pares de coordenadas).  
  
 Exemplo:  
 Input: {'people': [['Marina Sanches', ((10,20),(20,13)), 'Berlin'], ['Tiago Delgado', ((11,34),(14,23)), 'Brazil']]}  
 Pergunta: "Qual a posicao onde o nome da pessoa é Marina?"  
  
 Sua resposta deve ser apenas: ((10, 20), (20, 13))  
 Also make sure to put the name of the closest one. for example, if the user says 'mother' as the name, and there is a person named 'mother' and another named 'friend's mother' then the correct one is 'mother'""")  
# suchen\_app\_ki.  
def sendMessageOnWhatsApp(conversation\_name, messages):  
 """This functions params are 'conversation\_names' : str and 'messages' : list.  
 It sends the messages, in order, to the target conversation.   
 The messages cannot be sent by the AI pretending to be the user, when writing the message, make sure to specify that it's you writing, by saying "Hello, it's -Name- here! etcetera.   
 Make sure to write the messages normally, but please try to always write a lot of text instead of a little bit.  
 Usually prefer to separate the messages in a bunch of messages, instead of writing a single message. Please usually send a bunch of messages.  
 The function does not oppens the whatsappApp by itself. So make sure to open the app before calling this function.  
 Final reminder: This function does not send messages to multiple people at once, in case you want to send messages to more than one person, make sure to call it twice and set the parameters to the two different functions.  
 Returns if the message was sent or not."""  
 global suchen\_gesprach\_ki  
 screen = yf.getCurrentScreenImage()  
 predictions = yf.getModelPrediction("WhatsApp",screen)  
 # element\_classes = yf.getTextFromImage(screen, predictions)  
 if "search\_conversation\_bar" not in predictions.keys():  
 # print("No search bar found")  
 raise KeyError("No search bar found. This likely means that the app is not open. Make sure to open the app before calling this function.")  
 Search\_bar = predictions["search\_conversation\_bar"][0]  
 yf.clickOnElement(Search\_bar)  
 writeText(conversation\_name)  
 screen = yf.getCurrentScreenImage()  
  
 predictions = yf.getModelPrediction("WhatsApp",screen)  
 element\_classes = yf.getTextFromImage(screen, predictions)  
 people = element\_classes["conversation\_box"]  
  
 answer = suchen\_gesprach\_ki.message(f"""input: {people}   
 pergunta: Qual a posicao onde o nome da pessoa é {conversation\_name}?""")  
  
 # print(answer)  
  
 textList = list(answer)  
 count\_a = 0  
 count\_b = 0  
 for i in textList:  
 if i == "(":  
 count\_a += 1  
 if i == ")":  
 count\_b += 1  
   
 while count\_a > 3:  
 textList.remove("(")  
 count\_a-= 1  
   
 # while count\_b > 4:   
 # textList.remove(")")  
 # count\_b-= 1  
  
 anw = "".join(textList)  
  
 def extract\_tuple(text):  
 numbers = list(map(int, re.findall(r'\d+', text)))  
 if len(numbers) == 4:  
 return ((numbers[0], numbers[1]), (numbers[2], numbers[3]))  
 else:  
 print(text)  
 raise ValueError("Couldn't find exactly 4 numbers in the text. This likely means that the conversation name was not found. Please check the name and try again or tell the user.")  
  
 pos = extract\_tuple(anw)  
 # print(anw)  
 # input()  
 # pos = ast.literal\_eval(anw)  
 yf.clickOnElement(pos)  
 if "send\_message\_bar" not in predictions.keys():  
 predictions = yf.getModelPrediction("WhatsApp",screen)  
 if "send\_message\_bar" not in predictions.keys():  
 print("No send message area found")  
 raise KeyError("No send message area found. This likely means that the app is not open. Make sure to open the app before calling this function.")  
 send\_area = predictions["send\_message\_bar"][0]  
 # print(send\_area)  
 # input()  
 if isinstance(messages, list):  
 for message in messages:  
 yf.clickOnElement(send\_area)  
 writeText(message)  
 gui.press("enter")  
 time.sleep(0.25)  
 return "messages sent"  
   
 yf.clickOnElement(send\_area)  
 writeText(messages)  
 gui.press("enter")  
 time.sleep(0.25)  
 return "message sent"

### 3. Funções Matemáticas Básicas

def addNumbers(a, b):  
 """Params: a -> float and b -> float. This function takes two numbers and returns the sum of a + b"""  
 return a + b  
  
def multiplyNumbers(a, b):  
 """Params: a -> float and b -> float. This function takes two numbers and returns the product of a \* b"""  
 return a\*b

### 4. Escrita de Texto

def writeText(text):  
 """This function takes a string and writes it to the screen"""  
 gui.hotkey('command', 'a')  
 gui.hotkey('command', 'a')  
 time.sleep(0.2)  
 gui.typewrite(text)  
 time.sleep(0.5)  
 return None

## Integração de Visão Computacional

O PICO usa modelos YOLO para entender e interagir com interfaces de computador:

### Predição com Modelos YOLO

def getModelPrediction(app\_name : str, imagex : np.array, grayscale : bool = True):  
 """This function takes the ai model and the image and returns the predictions of the model"""  
 global models  
 image = copy.deepcopy(imagex)  
 if len(image.shape) == 2: # grayscale with shape (H, W)  
 image = cv2.cvtColor(image, cv2.COLOR\_GRAY2BGR)  
 if image.shape[2] == 4:  
 image = cv2.cvtColor(image, cv2.COLOR\_BGRA2BGR)  
   
 if grayscale:  
 if len(image.shape) == 3 and image.shape[2] == 3:  
 image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
   
 app\_name = app\_name.lower()  
 if app\_name != "find\_app" and app\_name != "separate\_text":  
 find\_preds = getModelPrediction("find\_app", image, True)  
 # print(find\_preds)  
 asa = "whatsapp" if app\_name == "whatsapp2" else app\_name  
 if asa not in find\_preds.keys():  
 print(f"App '{app\_name}' not found in predictions.")  
 raise KeyError(f"App '{app\_name}' not found in predictions.")  
  
 app\_area = find\_preds[asa][0]  
 x1, y1 = app\_area[0]  
 x2, y2 = app\_area[1]  
  
 # Create a black image  
 masked = np.zeros\_like(image)  
  
 # Copy only the app area from original image  
 if len(image.shape) == 2:  
 masked[y1:y2, x1:x2] = image[y1:y2, x1:x2] # grayscale  
 else:  
 masked[y1:y2, x1:x2, :] = image[y1:y2, x1:x2, :] # color  
  
 image = masked  
   
 if len(image.shape) == 2: # grayscale with shape (H, W)  
 image = cv2.cvtColor(image, cv2.COLOR\_GRAY2BGR)  
   
 model = models[app\_name]  
 results = model.predict(source=copy.deepcopy(image), save=False, conf=0.25, verbose=False)  
  
 objects = {}  
  
 r = results[0]  
 for box in r.boxes:  
 x1, y1, x2, y2 = map(int, box.xyxy[0].tolist())  
 conf = box.conf[0].item()  
 class\_id = int(box.cls[0].item())  
 label = r.names[class\_id]  
  
 if label not in objects.keys():  
 objects[label] = []  
 objects[label].append(((x1,y1),(x2,y2)))  
  
 objects = {k.lower(): v for k, v in objects.items()}  
  
 # saveImage(image, "teste.png", objects)  
  
 return objects

### Carregamento de Modelos

models = {}  
def load\_all\_models(models\_dir="../../yoloModels"):  
 """Preload all YOLO models from the given directory into the models dict."""  
 for folder in os.listdir(models\_dir):  
 model\_path = os.path.join(models\_dir, folder, "mainModel.pt")  
 if os.path.isfile(model\_path):  
 print(f"Loading model: {folder}")  
 models[folder.lower()] = YOLO(model\_path)  
 print(f"Loaded {len(models)} models.")  
  
# Call this once at startup  
load\_all\_models()  
DEVICE = "mps" if torch.backends.mps.is\_available() else "cpu"  
print(DEVICE)  
  
# one-time: warmup (avoids first-call penalty)  
for m in models.values():  
 m.to(DEVICE)  
 \_ = m.predict(source=np.zeros((640,640,3), dtype=np.uint8), imgsz=640, device=DEVICE, verbose=False)

### Ações no Computador

def clickOnElement(element):  
 """This function takes a tuple with the coordinates of the element and clicks on it"""  
 global proportions  
 # proportions = proportions  
 if isinstance(element[1], str):  
 element = element[0]  
 middle\_point = getMiddlePosition(element)  
 x\_fake = middle\_point[0]  
 y\_fake = middle\_point[1]  
 x = x\_fake \* proportions[0]  
 y = y\_fake \* proportions[1]  
 gui.click(x, y)  
 # time.sleep(0.2)  
 gui.click(x, y)  
 # time.sleep(0.5)  
 return None  
  
def getMiddlePosition(rectangle\_array):  
 """This function takes the coordinates of a rectangle and returns the middle point of it"""  
   
 x\_ = (rectangle\_array[0][0] + rectangle\_array[1][0])/2  
 y\_ = (rectangle\_array[0][1] + rectangle\_array[1][1])/2  
  
 middle\_point = (x\_, y\_)  
  
 return middle\_point

## Saída Text-to-Speech

O PICO fornece feedback de áudio usando um sistema de speaker personalizado:

### Classe speakerThing

class speakerThing:  
 """  
 Threaded TTS helper.  
 - On macOS: uses the built-in `say` command (reliable from background threads).  
 - On Linux/Windows: uses pyttsx3.  
  
 API:  
 speak(text or [texts]) -> enqueue speech  
 stop() -> stop current utterance  
 flush\_queue() -> clear pending (not-yet-spoken) items  
 wait\_until\_done() -> block until queue drained & not speaking  
 set\_rate(rate) -> adjust speed (pyttsx3) / via `say -r` on macOS  
 set\_volume(volume) -> adjust volume (pyttsx3 only)  
 set\_voice(name\_or\_id) -> select voice (best-effort on macOS via -v)  
 list\_voices() -> list voices (pyttsx3 only)  
 shutdown() -> clean shutdown  
  
 Notes:  
 \* Use as a context manager to guarantee cleanup.  
 \* macOS volume control via `say` isn't directly supported; use system volume.  
 """  
   
 def \_\_init\_\_(self, rate: int = 180, volume: float = 0.9, debug: bool = False):  
 self.debug = debug  
 self.\_q = queue.SimpleQueue()  
 self.\_alive = threading.Event()  
 self.\_alive.set()  
  
 self.\_rate = int(rate)  
 self.\_volume = float(volume)  
 self.\_using\_mac\_say = platform.system() == "Darwin"  
   
 if not self.\_using\_mac\_say:  
 self.\_engine = pyttsx3.init()  
 self.\_engine.setProperty("rate", self.\_rate)  
 self.\_engine.setProperty("volume", max(0.0, min(1.0, self.\_volume)))  
  
 self.\_worker = threading.Thread(target=self.\_worker\_loop, daemon=True)  
 self.\_worker.start()  
   
 def speak(self, text: Union[str, Iterable[str]]) -> None:  
 if isinstance(text, str):  
 self.\_q.put(text)  
 else:  
 for t in text:  
 self.\_q.put(str(t))  
  
 def \_speak\_mac(self, text: str) -> None:  
 # Build args for `say`. Use -r for rate, -v for voice if provided.  
 args = ["say", "-r", str(self.\_rate)]  
 if self.\_voice:  
 args += ["-v", self.\_voice]  
 args.append(text)  
 # if self.debug:  
 # print(f"[speakerThing] macOS say: {' '.join(args)}")  
 # Run synchronously so queue order is preserved  
 subprocess.run(args, check=False)  
  
 def \_speak\_pyttsx3(self, text: str) -> None:  
 assert self.\_engine is not None  
 self.\_engine.say(text)  
 self.\_engine.runAndWait()

### Inicialização do Speaker

speaker = af.speakerThing(rate=180, volume=0.9, debug=True)  
  
yield user\_response  
speaker.speak(user\_response)  
speaker.speak(finalResponse)  
speaker.wait\_until\_done()

## Exemplo de Fluxo do Sistema

Aqui está como uma interação completa flui através do sistema:

### Cenário: Usuário diz “Oi PICO, você poderia enviar uma mensagem para a Marina dizendo olá no WhatsApp?”

**1. Vosk** converte fala em texto:

Entrada: "Oi PICO, você poderia enviar uma mensagem para a Marina dizendo olá no WhatsApp?"

**2. PICO AI** responde:

{  
 "user\_response": "Claro, vou abrir o WhatsApp e enviar uma mensagem dizendo olá!",  
 "prompt": "Abrir WhatsApp, enviar mensagens para Marina dizendo 'Olá, aqui é o PICO!' 'Como você está?'"  
}

**3. Sistema fala** resposta amigável ao usuário

**4. Consciousness AI** quebra isso em passos:

**Passo 1 - Verificar se WhatsApp está aberto:**

{"function": "checkIfAppIsOppened", "parameters": ["WhatsApp"]}

**Passo 2 - Abrir WhatsApp se necessário:**

{"function": "openApp", "parameters": ["WhatsApp", "mac"]}

**Passo 3 - Enviar mensagem:**

{"function": "sendMessageOnWhatsApp", "parameters": ["Marina", ["Olá, aqui é o PICO!", "Como você está?"]]}

**5. Sistema executa** cada função sequencialmente: - Abre WhatsApp usando Command+Space - Usa visão computacional para encontrar elementos da UI - Clica na barra de pesquisa - Digita “Marina” - Encontra a conversa usando IA - Clica na conversa - Digita e envia as mensagens

**6. PICO fala** confirmação final ao usuário:

"Pronto! Enviei as mensagens para a Marina no WhatsApp. As mensagens 'Olá, aqui é o PICO!' e 'Como você está?' foram enviadas com sucesso."

## Pontos de Integração do Sistema

O sistema inclui múltiplos pontos de integração para diferentes casos de uso:

### 1. API Flask para Integração Web

from flask import Flask, jsonify, request  
from flask\_cors import CORS  
  
app = Flask(\_\_name\_\_)  
CORS(app)  
  
@app.route("/pico/prompt", methods=["POST", "GET"])  
def hi():  
 data = request.get\_json(force=True)  
 prompt = data["prompt"]  
 response = pico\_main.sendUserInput(prompt)  
 print(response)  
 return jsonify(response)  
  
@app.route("/send\_message", methods=["POST"])  
def send\_message():  
 data = request.json()  
 user\_message = data["message"]  
 pico\_main.sendUserInput()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(debug=True)

### 2. Interface STDIO para Integração Desktop

import sys  
import json  
from pico\_main import sendUserInput  
  
def main():  
 for raw\_line in sys.stdin:  
 line = raw\_line.strip()  
 if not line:  
 continue  
 try:  
 request = json.loads(line)  
 request\_id = request.get("id")  
 prompt = request.get("prompt", "")  
 speak\_out\_loud = bool(request.get("speak", True))  
  
 text\_response = sendUserInput(prompt, 0, speak\_out\_loud)  
  
 response = {  
 "ok": True,  
 "id": request\_id,  
 "text": text\_response,  
 }  
 except Exception as e:  
 response = {  
 "ok": False,  
 "id": request.get("id") if isinstance(request, dict) else None,  
 "error": str(e),  
 }  
  
 sys.stdout.write(json.dumps(response) + "\n")  
 sys.stdout.flush()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

### 3. Interface de Linha de Comando Direta

# Uso direto no terminal para testes  
if \_\_name\_\_ == "\_\_main\_\_":  
 while True:  
 r = input("--> ")  
 time1 = time.time()  
 print()  
 for resp in getPicoResponse(r):  
 print(resp)  
 print()  
 time2 = time.time()  
 dt = time2 - time1  
 print(f"Tempo de resposta: {dt} segundos")  
 print(f"Memória PICO AI: {len(PICO\_AI.memory)} mensagens")

## Características Técnicas

### Modelos Utilizados

* **Speech-to-Text:** Vosk (modelos offline em português e inglês)
* **IA Principal:** GPT-4.1-nano e GPT-5-nano (OpenAI)
* **Visão Computacional:** YOLO (modelos personalizados treinados)
* **Text-to-Speech:** pyttsx3 (Windows/Linux) e comando ‘say’ (macOS)

### Dependências Principais

# Principais bibliotecas utilizadas  
import sounddevice as sd # Captura de áudio  
import vosk # Reconhecimento de fala  
import cv2 # Processamento de imagem  
import numpy as np # Manipulação de arrays  
import pyautogui as gui # Automação de interface  
import pytesseract # OCR para texto  
import pyttsx3 # Text-to-speech  
from ultralytics import YOLO # Modelos de visão computacional  
from langchain\_community.chat\_models import ChatOllama  
from langchain.chat\_models import ChatOpenAI

### Estrutura de Arquivos

PICO\_AI\_Instructions/  
├── picoInstructions.txt # Instruções da personalidade  
├── consiousnessInstructions.txt # Instruções de execução  
└── aiPersonality.txt # Personalidade e informações  
  
src/backend/  
├── pico\_main.py # Orquestrador principal  
├── chatFunctions.py # Funções de chat e computador  
├── voskThing.py # Interface Vosk  
├── audioFunctions.py # Sistema TTS  
├── yoloFunctions.py # Visão computacional  
├── pico\_resolver.py # API Flask  
└── pico\_stdio.py # Interface STDIO  
  
yoloModels/  
├── find\_app/ # Modelo para detectar apps  
├── whatsapp/ # Modelo específico WhatsApp  
├── separate\_text/ # Modelo para separar texto  
└── spotify/ # Modelo específico Spotify  
  
voskModels/  
└── English/ # Modelo de reconhecimento de fala

## Conclusão

O sistema P.I.C.O. representa uma solução inovadora para controle de computador por voz, combinando tecnologias avançadas de IA, processamento de fala e visão computacional. A arquitetura modular permite fácil extensão com novas funcionalidades e integração com diferentes plataformas.

### Principais Vantagens:

* **Funcionamento Offline:** Não depende de conexão com internet
* **Interface Natural:** Comandos em linguagem natural
* **Visão Computacional:** Entende interfaces gráficas
* **Modular:** Fácil adição de novas funções
* **Multiplataforma:** Funciona em Windows, macOS e Linux
* **Personalizável:** Personalidade e comportamento adaptáveis

### Casos de Uso:

* Controle de aplicativos por voz
* Automação de tarefas repetitivas
* Assistência para pessoas com dificuldades motoras
* Integração com sistemas domésticos inteligentes
* Interface para aplicações empresariais

O P.I.C.O. demonstra o potencial das tecnologias de IA conversacional aplicadas ao controle de computadores, oferecendo uma experiência mais natural e eficiente para os usuários.