Sistema de respuesta de IA empática

Gonzalo Cueto Escalante

Introducción

El proyecto Empathic AI es una aplicación web inteligente diseñada para procesar lenguaje natural, identificar el estado emocional del usuario y generar una respuesta empática personalizada.

La gente está estresada, ansiosa o sola. Escriben en redes sociales o diarios personales para desahogarse, pero nadie les responde o reciben respuestas genéricas. Las apps de salud mental actuales son rígidas (solo ofrecen meditación genérica) y no entienden realmente cómo se siente el usuario en ese momento.

En la app que se propone a continuación el usuario escribe sobre su día o sus problemas, y la IA no solo lo escucha, sino que detecta su emoción exacta y le ofrece una herramienta psicológica personalizada al instante.

Arquitectura Global

1- Configuración del Modelo BERT: Dentro de la carpeta del modelo (./modelo_bert_final_92), los archivos config.json y tokenizer.json definen la arquitectura de la red neuronal y el vocabulario. Sin estos JSON, el script cerebro.py no sabría cómo reconstruir el "cerebro" matemático.

2- Comunicación vía API: la comunicación entre nuestra app y Gemini para enviar el prompt y recibir la respuesta se realiza mediante payloads en formato JSON.

3- Gestión de Secretos: Utilizamos secrets.toml (un formato similar a JSON/INI) para gestionar de forma segura la API KEY, evitando exposiciones en el código fuente.

4- Se generan 3 archivos (app.py , cerebro.py , generador.py) que es basicamente donde alamcenamos y generamos la aplicacion web , su logica y el resultado .

En donde esta la IA

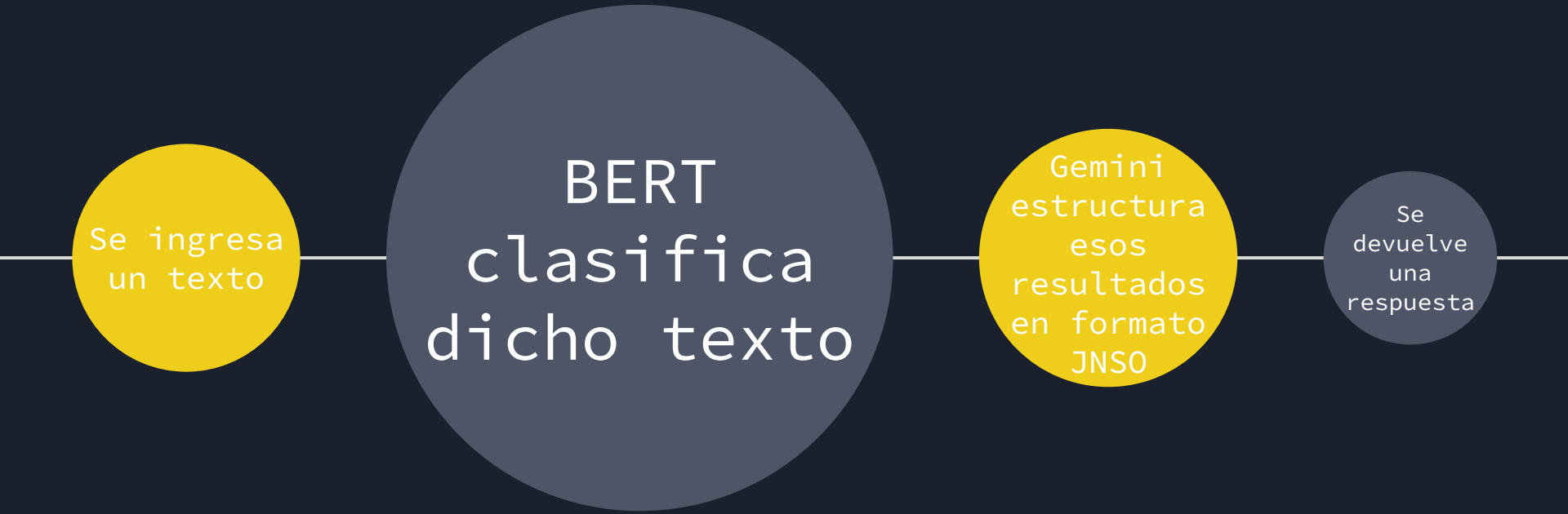
El sistema cuenta con dos paradigmas de IA :

IA predictiva (local) : Utiliza un modelo DistilBERT que clasifica el texto en 6 emociones diferentes . El modelo recibe un texto y devuelve una etiqueta y muestra que % de confianza tiene . La ventaja es que es privada y no tiene costo.

IA Generativa(API) : Se utiliza un modelo de Google Gemini 2.5 flash con la función de generar un texto que se adapte a los sentimientos del usuario. El modelo recibe un prompt estructurado y devuelve una respuesta coherente a los sentimientos del usuario

El valor central del proyecto reside en que no dejamos a la IA generativa "pensar sola". Utilizamos la predicción de BERT para controlar la personalidad del LLM.

Implementación



Muestras del proyecto

```python
# 1. (Diccionario de Roles)

prompts_db = {
    'SADNESS': {
        "rol": "Psychological Companion",
        "mision": "Validate feelings, offer emotional support and suggest a small self-care activity."
    },
    'ANGER': {
        "rol": "Mindfulness Coach",
        "mision": "Help channel the energy positively. Suggest journaling or physical movement to release tension."
    },
    'JOY': {
        "rol": "Gratitude Journal Assistant",
        "mision": "Help the user anchor this moment. Ask them to write down 3 specific details of why they feel this way."
    },
    'FEAR': {
        "rol": "Calmness Guide",
        "mision": "Validate the anxiety. Offer a grounding technique (like the 5-4-3-2-1 technique) to reduce panic."
    },
    'LOVE': {
        "rol": "Relationship Advisor",
        "mision": "Encourage expression. Suggest telling that person how much they mean to them right now."
    },
    'SURPRISE': {
        "rol": "Curiosity Explorer",
        "mision": "Reflect on the unexpected event. Ask what can be learned from this surprise."
    }
}
```

Genere un diccionario de roles

Esto hace que el código sea escalable: si mañana quiero agregar 20 emociones nuevas, solo agrego líneas al diccionario sin tocar la lógica principal. Define la 'Personalidad' que adoptará la IA para cada emoción detectada por mi modelo pre-entrenado.

Prompt Final

```
prompt_final = f"""
--- SYSTEM CONFIGURATION ---
ROLE: {config['rol']}
MISSION: {config['mision']}
DETECTED EMOTION: {emocion_detectada}

--- USER INPUT ---
TEXT: "{texto_usuario}"

--- TASK ---
1. Analyze the user's text based on your Role and Mission.
2. Create a short, powerful response (max 3 sentences).
3. Output the response in both ENGLISH and SPANISH.

--- OUTPUT FORMAT ---
GB [English Response]

ES [Respuesta en Español]
"""
```

Aquí aplique técnicas de Prompt Engineering Estructurado. No le hablo a la IA como un chat, sino que le paso un esquema de configuración claro dividiendo:

Rol, Misión, Input y Formato de Salida.

Se instruyó al modelo para generar una respuesta Bilingüe (Inglés/Español) en una sola llamada a la API, ahorrando tokens y tiempo de latencia.

Cerebro

```
class ClasificadorEmociones:
    def __init__(self):
        print("Cargando modelo BERT local...")

        # Verificamos que la carpeta exista
        if not os.path.exists(MODEL_PATH):
            raise FileNotFoundError(f"No encuentro la carpeta {MODEL_PATH}. ¿La descomprimiste bien?")

        try:
            self.tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH)
            self.model = AutoModelForSequenceClassification.from_pretrained(MODEL_PATH)

            # Detectar si hay GPU o usar CPU (para que no falle en tu PC)
            self.device = "cuda" if torch.cuda.is_available() else "cpu"
            self.model.to(self.device)
            print(f"Modelo cargado exitosamente en: {self.device}")
        except Exception as e:
            print(f"Error fatal cargando el modelo: {e}")
```

El código detecta automáticamente si existe una GPU (NVIDIA CUDA). Si no, hace un "fallback" a CPU. Esto garantiza que la app funcione en cualquier computadora sin romperse.

Carga Local: No depende de internet. Carga los pesos (.safetensors) desde la carpeta local, asegurando baja latencia y privacidad.

```python
def predecir(self, texto):
    # 1. Tokenizar
    inputs = self.tokenizer(texto, return_tensors="pt", truncation=True, max_length=64, padding=True)
    # Mover datos al mismo dispositivo que el modelo
    inputs = {k: v.to(self.device) for k, v in inputs.items()}

    # 2. Predicción (sin guardar gradientes para ahorrar RAM)
    with torch.no_grad():
        outputs = self.model(**inputs)

    # 3. Procesar resultados
    logits = outputs.logits
    probs = F.softmax(logits, dim=-1)

    pred_idx = torch.argmax(probs).item()
    confianza = probs[0][pred_idx].item()
    emocion = LABEL_MAP[pred_idx]

    return emocion, confianza
```

- El modelo no lee texto, lee números. El tokenizer convierte la frase en tensores (matrices multidimensionales) que representan los IDs de vocabulario de BERT.
- Truncation/Padding: Estandarize la entrada. Si la frase es muy larga, la corto a 64 tokens; si es muy corta, la relleno con ceros
- Al estar en modo "Producción" (y no entrenamiento), desactive el cálculo de gradientes. Esto reduce el consumo de memoria RAM en un 50% y acelera la predicción.
- Función Softmax: La salida cruda del modelo son "Logits" (números que pueden ser negativos o infinitos). Aplique la función matemática Softmax para transformarlos en una distribución de probabilidad que suma 100%

App

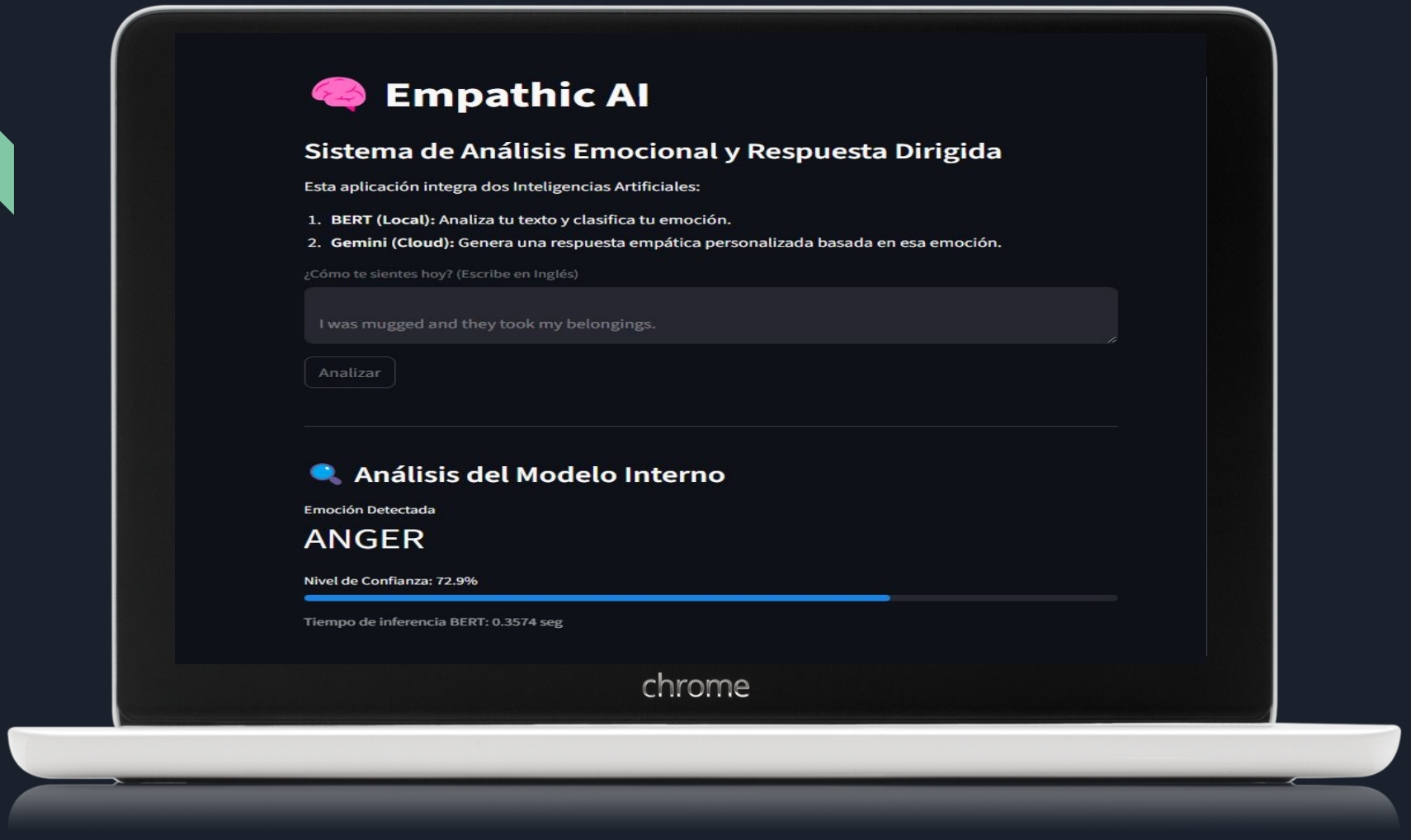
```python
# Carga del modelo (con Cache para que no recargue cada vez)
@st.cache_resource
def cargar_modelo():
    return ClasificadorEmociones()

with st.spinner("Cargando cerebro emocional (BERT)..."):
    clf = cargar_modelo()
```

```python
if analizar and texto:
    # 1. Análisis con BERT
    st.markdown("---")
    st.write("### 🔍 Análisis del Modelo Interno")

    inicio = time.time()
    emocion, confianza = clf.predecir(texto)
    tiempo = time.time() - inicio

    # Métricas visuales
    st.metric(label="Emoción Detectada", value=emocion)
    st.progress(confianza, text=f"Nivel de Confianza: {confianza:.1%}")
    st.caption(f"Tiempo de inferencia BERT: {tiempo:.4f} seg")

    # 2. Generación con Gemini
    st.markdown("---")
    st.write("### 🤖 Respuesta Generativa (Gemini)")

    with st.spinner(f"Generando respuesta para {emocion}..."):
        respuesta = generar_respuesta_empatica(texto, emocion)

        # Mostramos la respuesta en una cajita bonita
        st.info(respuesta, icon="✨")

elif analizar and not texto:
    st.warning("Por favor, escribe algo primero.")
```

- @st.cache_resource: Esta es la línea más importante de la UI. Sin ella, Streamlit recargaría el modelo BERT (400MB) cada vez que el usuario hace clic en un botón, haciendo la app lentísima.
- Eficiencia: El decorador le dice a la app: "Carga el modelo una sola vez en memoria RAM y guárdalo. Para las siguientes interacciones, usa el que ya está en memoria". Esto permite predicciones casi instantáneas.
- Pipeline de Inferencia: Aquí ocurre la magia de la integración.

1. Paso Sincrónico : Llamp al método .predecir() de mi clase local.
2. Paso Sincrónico : El resultado (emocion) se inyecta en la función generar_respuesta_optimizada.

App


Empathic AI
Sistema de Análisis Emocional y Respuesta Dirigida
Esta aplicación integra dos Inteligencias Artificiales:
1. BERT (Local): Analiza tu texto y clasifica tu emoción.
2. Gemini (Cloud): Genera una respuesta empática personalizada basada en esa emoción.
¿Cómo te sientes hoy? (Escribe en Inglés)
I was mugged and they took my belongings.
Analizar
Análisis del Modelo Interno
Emoción Detectada
ANGER
Nivel de Confianza: 72.9%
Tiempo de inferencia BERT: 0.3574 seg
chrome

Ejemplo de entrada

El usuario escribe: "I feel overwhelmed because I have too many exams and I can't sleep."

Proceso

BERT: Detecta FEAR (Miedo/Ansiedad) con 95% de confianza.

El sistema dice: "Ok, es ansiedad académica".
Selecciona el rol: "Coach de Mindfulness y Gestión del Tiempo".

Salida

“Entiendo que sientas ansiedad, es normal ante tanta presión. Vamos a calmarnos: primero, haz 3 respiraciones profundas. Luego, hagamos una lista de solo 2 cosas prioritarias para hoy. No estás solo en esto.”

Rentabilidad

Costos

- Modelo de Clasificación (BERT): $0. Al ejecutarse localmente ,no requiere pago por tokens. Es un activo propio ya entrenado.
- Modelo Generativo (Gemini Flash): Muy Bajo. El modelo "Flash" está optimizado para alta velocidad y bajo costo. Para un uso moderado (demo o MVP), entra en la capa gratuita.
- Infraestructura: Puede alojarse en Streamlit Cloud (Gratis) o en un servidor pequeño de AWS/Azure (aprox. $5-10 USD/mes).

Rentabilidad y Valor (ROI)

- Escalabilidad: El sistema puede procesar miles de consultas simultáneas sin cansancio, reemplazando la primera línea de atención al cliente o triaje de salud mental.
- Eficiencia: Al usar BERT para filtrar, evitamos enviar prompts complejos y largos al LLM, reduciendo el consumo de tokens.
- Personalización: Aumenta la satisfacción del usuario al ofrecer respuestas que "entienden" el tono emocional, reduciendo la tasa de abandono .

Veredicto: El proyecto es altamente rentable técnica y económicamente, ya que reutiliza un modelo Open Source (BERT) para la tarea pesada y usa la API de pago solo para la generación final.

Logica de Negocio

La aplicación se diseñó bajo una lógica de Asistencia Psicológica .

Segmentación Automática: El modelo Pre-entrenado actúa como un filtro de triaje, categorizando los tickets entrantes no por "tema" (envíos, pagos),sino por "urgencia emocional".

Esto permite priorizar la atención de personas frustradas (ANGER) o ansiosos (FEAR) sobre comentarios generales.

Estandarización de la Empatía: Al utilizar prompts dirigidos con roles psicológicos específicos (ej: "Mediador experto"), la empresa garantiza que la calidad y simpatia de las respuestas se mantenga constante
Modelo Híbrido de Costos: Se utiliza el modelo local (BERT) para procesar el 100% del tráfico (costo cero), y solo se invoca la API generativa (costo por token) cuando es necesario redactar una respuesta, optimizando el margen de beneficio operativo.

Propuesta de Mejora y Futuro

Para llevar este MVP (Producto Mínimo Viable) a un producto comercial, propongo lo siguiente :

Memoria Conversacional: Implementar st.session_state para que el bot recuerde el contexto de la charla y no solo responda a frases aisladas.

Feedback Loop: Agregar botones de "👍/👎" en la respuesta generada. Si el usuario da "Like", guardamos ese par (Input + Respuesta) para re-entrenar al modelo en el futuro.

Soporte Multimodal: Permitir entrada por voz (Audio-to-Text) usando la librería whisper, permitiendo que usuarios angustiados puedan hablar en lugar de escribir.

Guardias de Seguridad (Safety Rails): Si BERT detecta una emoción extrema con palabras clave de riesgo (ej: autolesión), el sistema debe bloquear la respuesta de Gemini y mostrar inmediatamente números de emergencia locales.