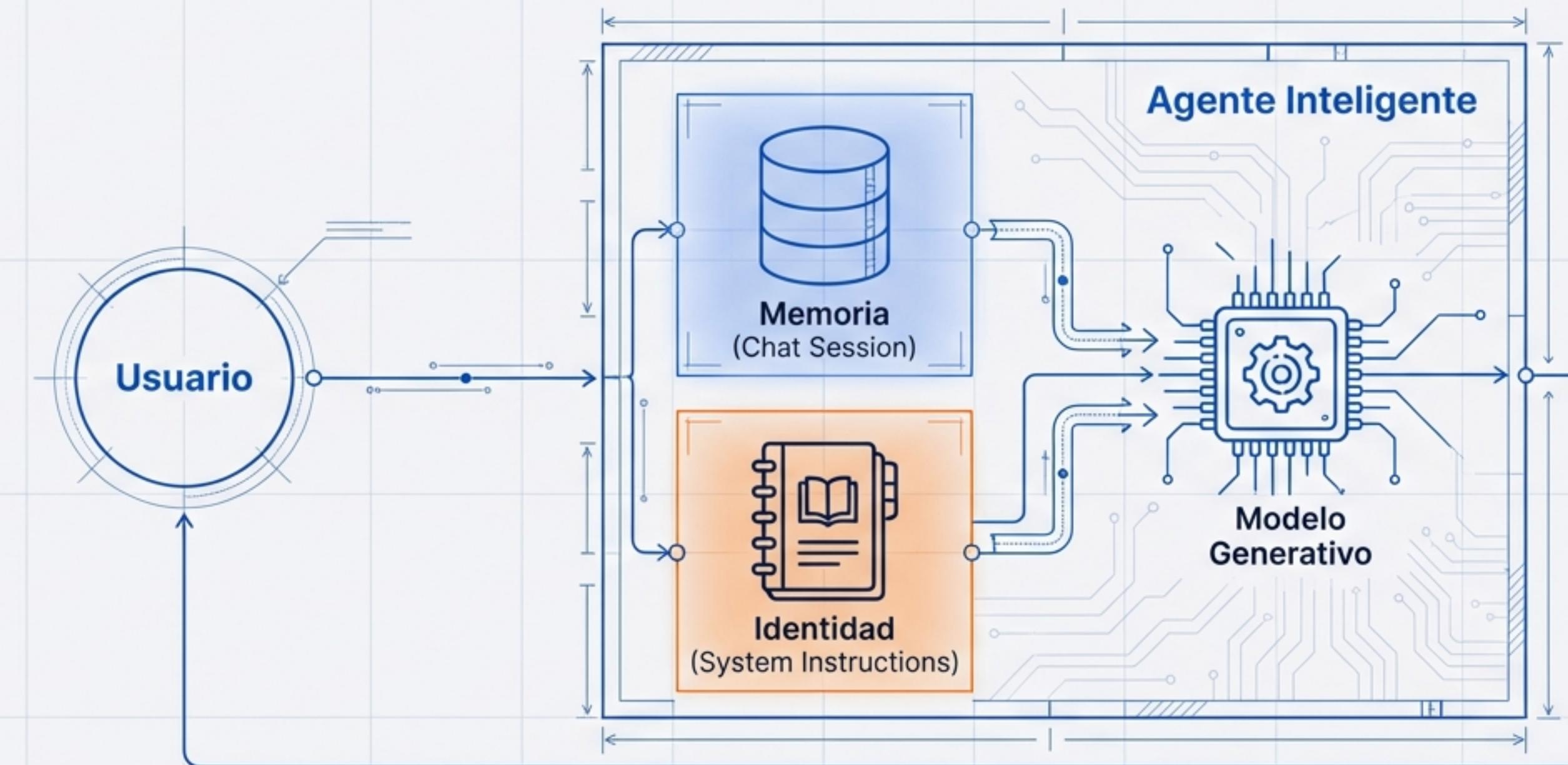


# Arquitectura de Agentes Inteligentes: Memoia y Personalidad

De la generación de texto simple a la construcción de socios conversacionales robustos.

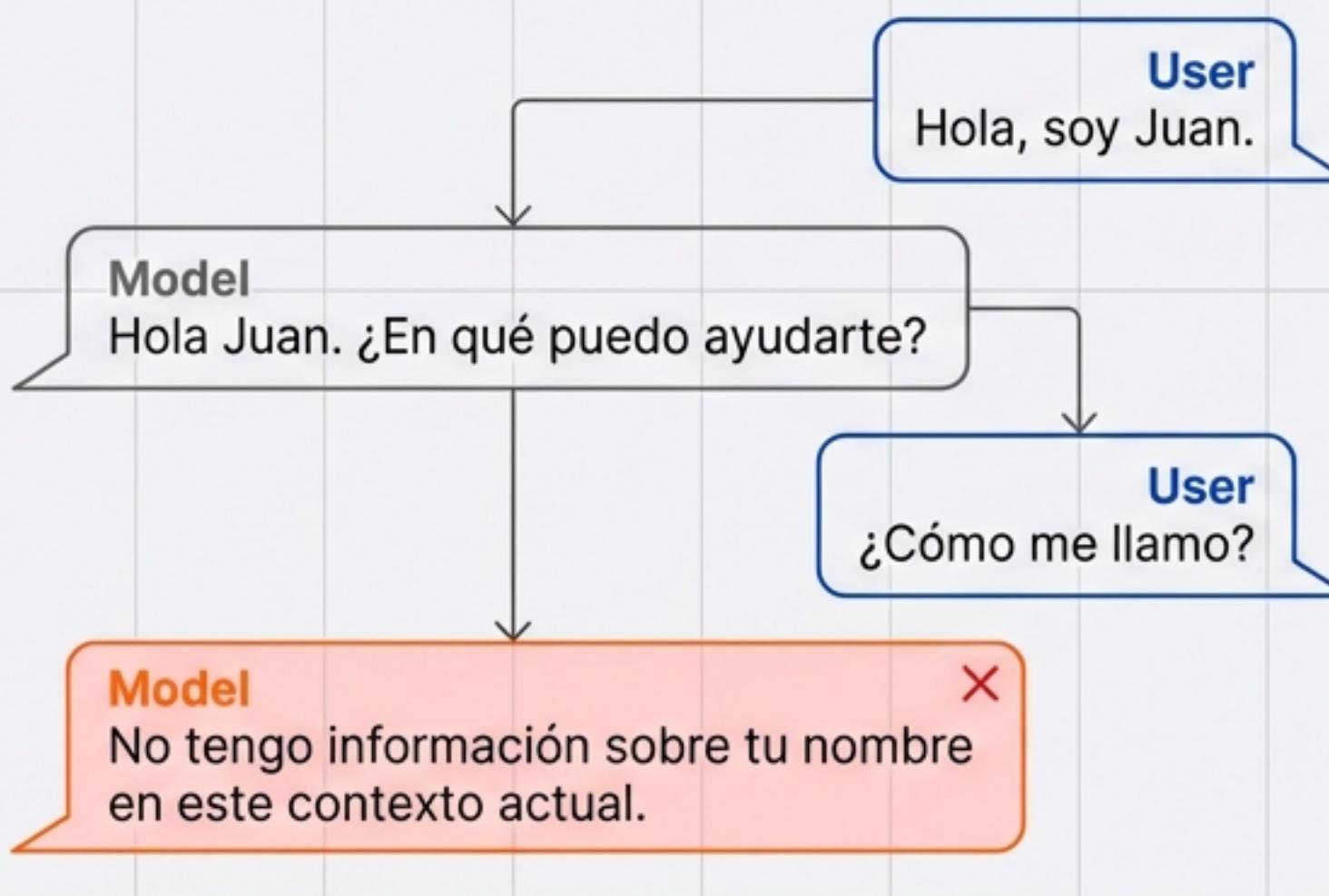


Prerrequisitos: Comprensión básica de generate\_content. Nivel: Intermedio.

# Superando la barrera "Stateless": El reto de la continuidad

## El Enfoque Stateless

Las llamadas básicas a `generate\_content` son islas aisladas. El modelo olvida el turno anterior inmediatamente.



## La Solución: Chat Sessions

Los **Chat Sessions** automatizan la gestión del historial, permitiendo conversaciones "multi-turno" coherentes.



**Sin persistencia, no hay conversación. Sin conversación, no hay agente.**

# Anatomía de una Chat Session

El mecanismo interno de la memoria a corto plazo.

## Historial Automático

El objeto `chat` encapsula el estado de la conversación (lista de mensajes) automáticamente. No es necesario concatenar strings manualmente.

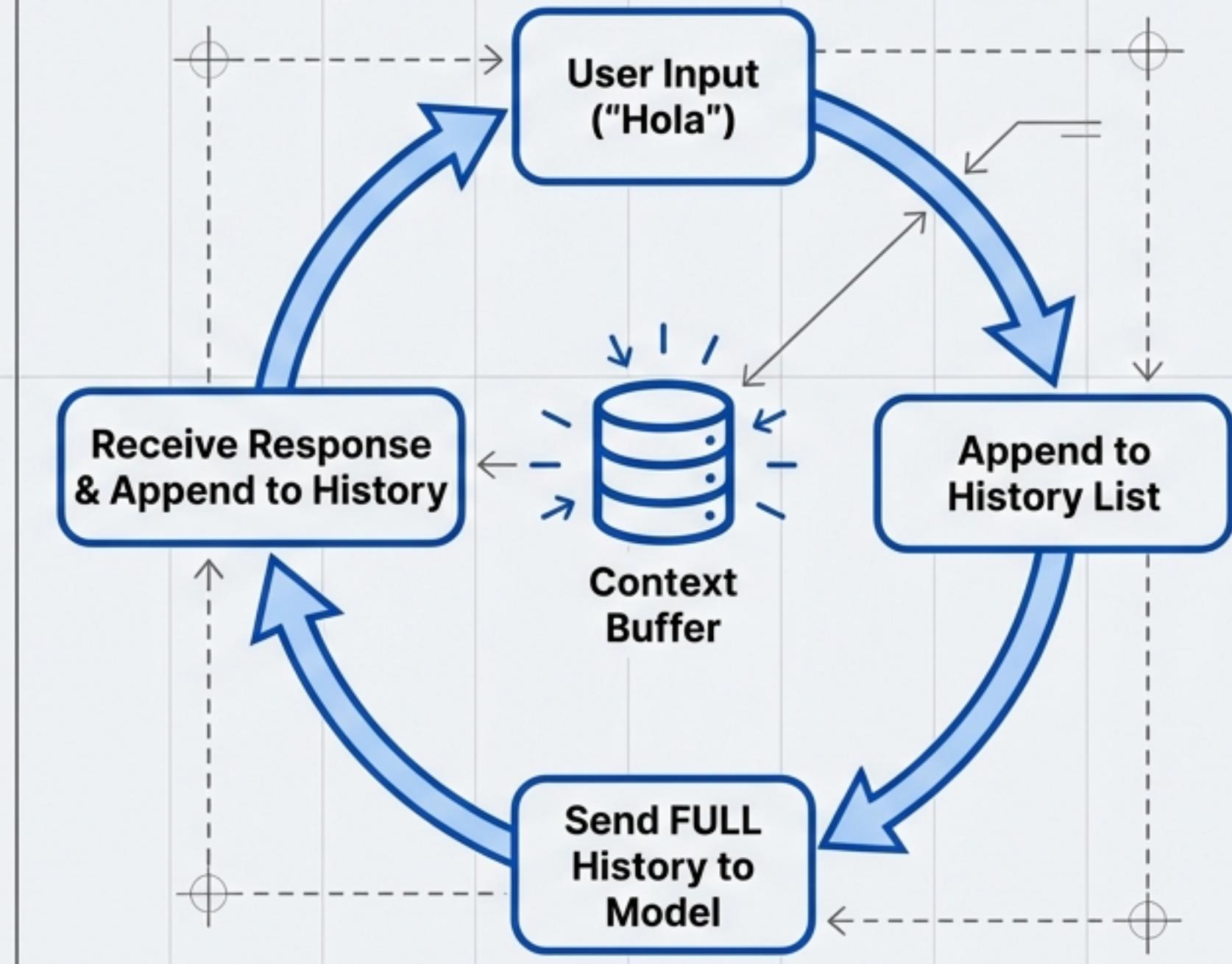
## Inicialización

Se puede iniciar desde cero (vacío) o con un `history` pre-cargado para restaurar sesiones pasadas.

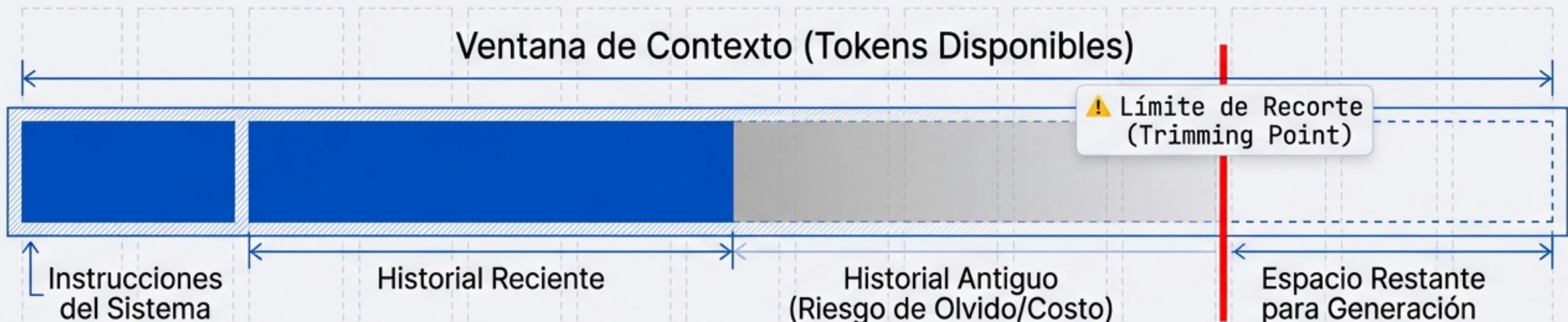
```
# Iniciar chat (puede incluir historial previo)
chat = model.start_chat(history=[...])

# Enviar mensaje (actualiza historial auto.)
response = chat.send_message("Hola")

# Inspeccionar la memoria
print(chat.history)
// Contiene el intercambio completo
```



# Ingeniería de Contexto: Gestión de Tokens y Límites



## \* El Costo del Infinito \*

El historial crece linealmente con cada turno. Un historial infinito desborda la ventana de contexto y aumenta la latencia.

## Monitorización

Inspeccionar `chat.history` regularmente es vital para entender el consumo de tokens.

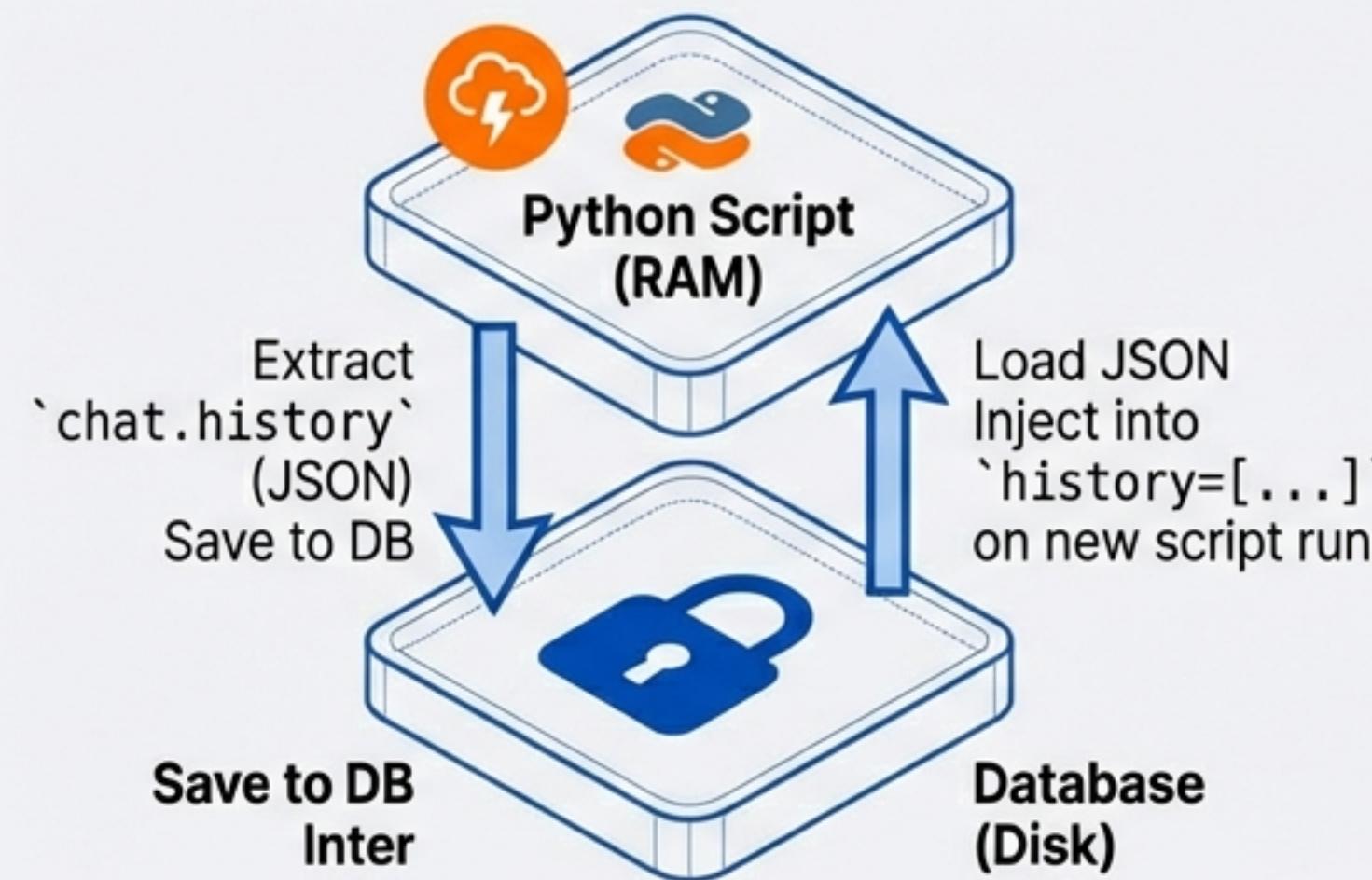
## Estrategia de Recorte

Implementar lógica para eliminar los mensajes más antiguos (FIFO) cuando se acerca al límite, preservando las instrucciones del sistema.



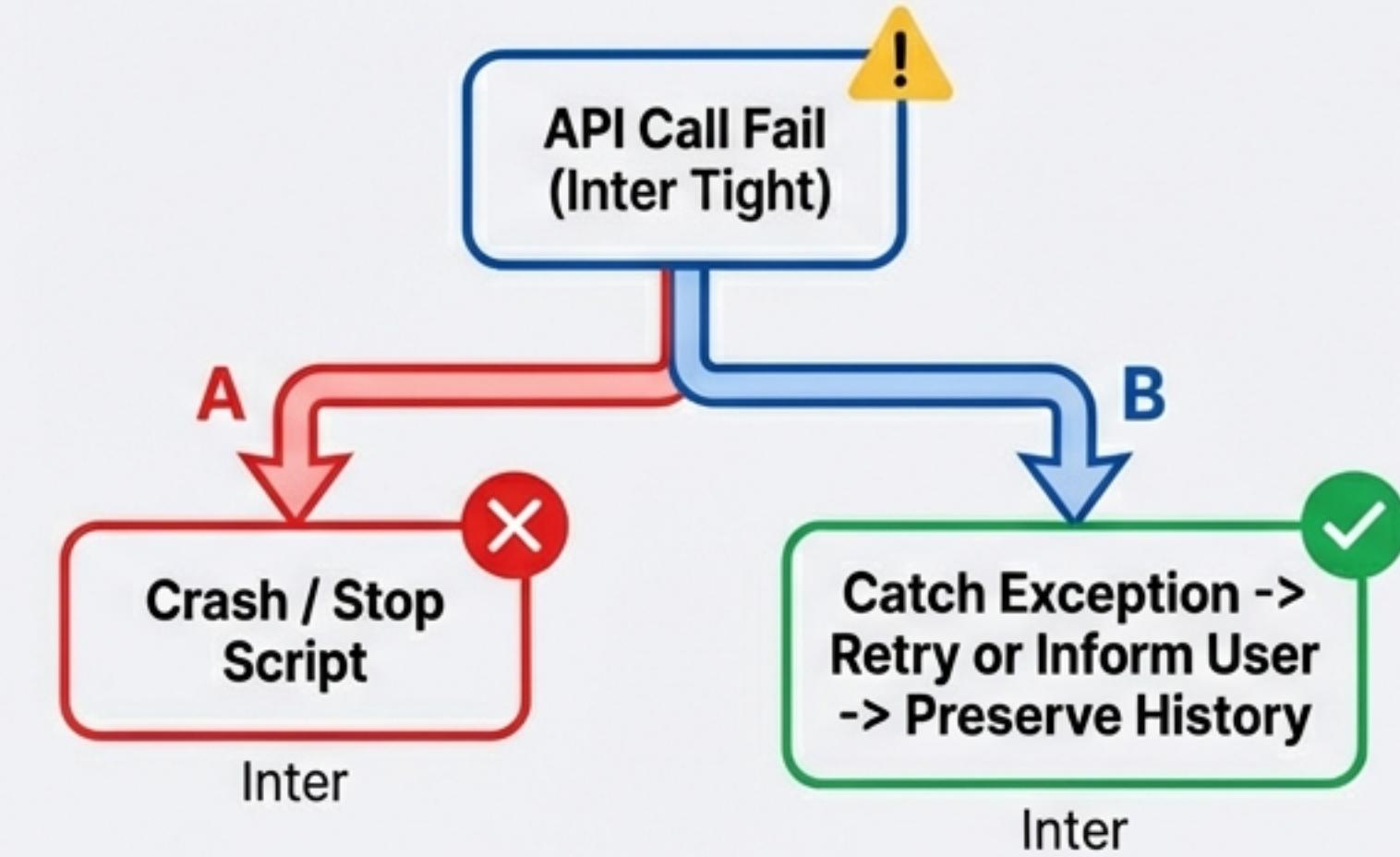
# Del Script a Producción: Persistencia y Robustez

## Persistencia (Base de Datos)



La memoria del script es volátil. Para sesiones largas, el estado debe guardarse externamente.

## Manejo de Errores



El objeto chat debe ser resiliente. Si una llamada falla, el historial no debe corromperse.

# Experiencia de Usuario: Streaming en Tiempo Real

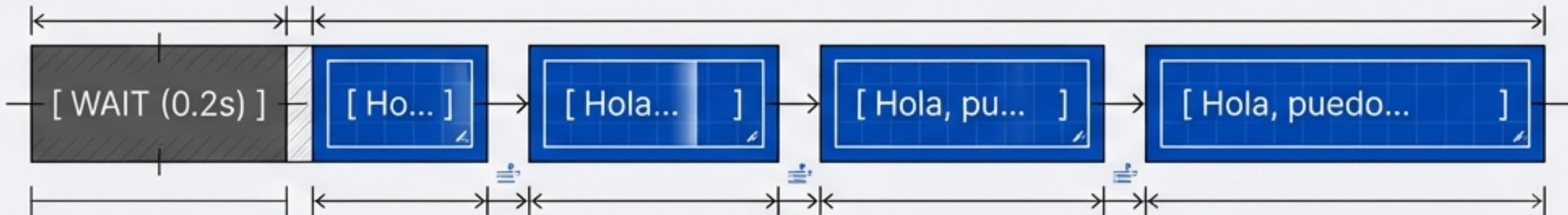
Reduciendo la latencia percibida (Time to First Token).

## Standard Generation (Blocking)



Alta latencia percibida. El usuario espera en silencio.

## Streaming Generation (Chunking)



Feedback inmediato. Sensación de pensamiento fluido.

### Por qué usar Streaming:

- 1. Mejora la UX en interfaces de chat.
- 2. Permite procesar la respuesta mientras se genera.
- 3. Estándar de la industria para LLMs.

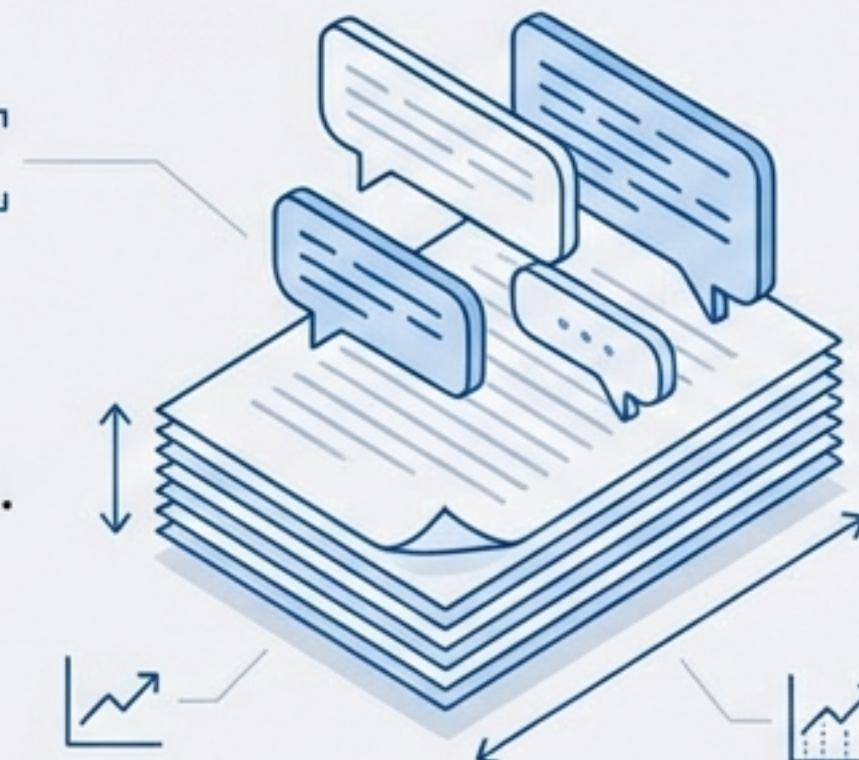
# Más allá de la Memoria: Definiendo la Identidad

System Instructions como el "Alma" del Agente.

## Chat History

### La Minuta de la Reunión

Registra qué se dijo. Es variable, crece con el tiempo y es circunstancial.



## System Instructions

### El Manual del Empleado

Define quién es el agente. Es inmutable, persistente y aplica a cada interacción.



**Diferenciador Clave:** A diferencia de un prompt de usuario, las System Instructions pesan sobre **todas** las respuestas y configuran el comportamiento base antes de la primera palabra.

# Jerarquía de Control y Seguridad

Estructurando la Prioridad de Instrucciones en LLMs.

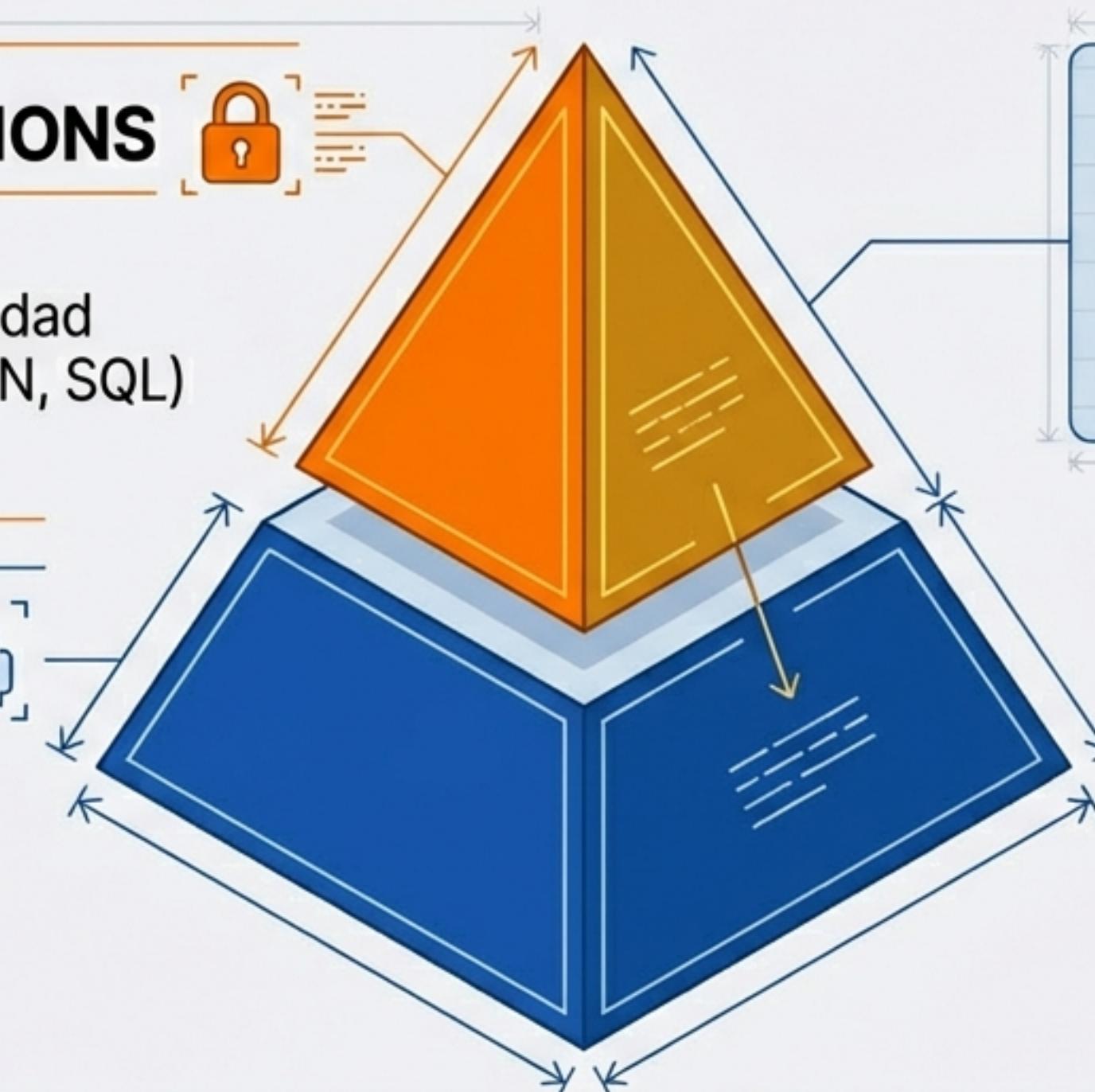
## SYSTEM INSTRUCTIONS

- Identidad y Rol
- Restricciones de Seguridad
- Formato de Salida (JSON, SQL)
- Tono y Estilo



## USER PROMPTS

- Preguntas del usuario
- Comandos inmediatos
- Datos de entrada



### Constitución del Agente:

Las instrucciones aquí tienen prioridad lógica. Ideales para evitar "jailbreaks" o salidas fuera de formato.

# Patrones de Diseño (I): Especialistas y Formateadores

Ejemplos prácticos de System Instructions para roles específicos y formatos de salida.

## El Asistente Especializado

### Input (System Instruction):

"Eres un experto en ciberseguridad. Usas terminología técnica precisa y priorizas la defensa."

### Output (Chat Simulation):

User: "¿Es segura mi contraseña?"



Agent: "Desde una perspectiva de entropía, una contraseña de 8 caracteres es vulnerable a ataques de fuerza bruta. Recomiendo implementar 2FA..." (Tone: Formal, Technical).

## El Formateador (Machine-to-Machine)

### Input (System Instruction):

"Tu única salida debe ser código SQL válido basado en la petición. Sin explicaciones."

### Output (Chat Simulation):



User: "Dame todos los usuarios activos de ayer."

Agent: `SELECT * FROM users WHERE status = 'active' AND date = '2023-10-27';`  
(Format: Pure Code).

# Patrones de Diseño (II): Moderadores y Herramientas

Ejemplos prácticos de System Instructions para roles de clasificación y uso de herramientas externas.

## El Moderador (Clasificador)

Input (System Instruction):

"Evalúa el sentimiento. Si es tóxico responde 'FLAG'. Si es seguro responde 'PASS'."



Output (Chat Simulation):



Odio este servicio, son unos inútiles.

FLAG



¿Cómo actualizo mi perfil?

PASS

## El Agente con Herramientas

Input (System Instruction):

"Si falta información para responder, pide buscar en la base de datos de inventario."



Output (Chat Simulation):



¿Tenemos stock de zapatos rojos?



Necesito consultar el inventario actual. Iniciando búsqueda en base de datos...

# La Síntesis: Integrando Identidad y Memoria

Código de implementación completo.

```
# 1. Definir la Identidad (System Instruction)
model = GenerativeModel(
    model_name="gemini-pro",
    system_instruction="Eres un tutor socrático. Nunca des la
                        respuesta directa; haz preguntas guía.")

# 2. Iniciar la Memoria (Chat Session)
chat = model.start_chat(history=[])

# 3. Interacción Viva
response = chat.send_message("¿Cuánto es 2 + 2?

print(response.text)
# Resultado: "¿Qué sucede si sumas dos unidades a otras dos?"
```

**Entidad Completa:**  Instrucción + Historial  
= Agente.

# Mejores Prácticas para Construcción de Agentes

Guía esencial para asegurar la robustez, claridad y eficacia de los agentes de IA



Checklist

## Especificidad en el Rol

Evitar ambigüedades. No diga “sé útil”, diga “eres un asistente de soporte técnico nivel 2”.



Checklist

## Validación Adversaria

Pruebe las System Instructions con “prompts adversarios” (intentos de romper el personaje) para asegurar robustez.



Checklist

## Gestión de Contexto

Implemente una lógica de recorte (trimming) para mantener el historial dentro de los límites de tokens sin perder la instrucción del sistema.



Checklist

## Formato Explícito

Si requiere JSON o Markdown, defina la estructura exacta en la System Instruction, no en el chat.

“Un buen agente es la suma de instrucciones claras y una memoria bien gestionada.”

# El Camino hacia la Ingeniería Avanzada

