

## Chatbot Inteligente de Flora y Fauna de Bolivia

Versión: 1.0.0

Tecnología: Python 3 / Tkinter

Tipo de Sistema: Asistente Virtual de Escritorio con Recuperación de Información en Tiempo Real (RAG Básico).

### 1. RESUMEN EJECUTIVO

El **Chatbot de Biodiversidad de Bolivia** es una aplicación de escritorio diseñada para democratizar el acceso a la información sobre la flora y fauna boliviana. A diferencia de los chatbots tradicionales que dependen únicamente de bases de datos estáticas, este sistema implementa una arquitectura híbrida: posee conocimientos precargados (modo offline) y capacidades de **Web Scraping** (modo online) para extraer información actualizada de fuentes confiables como Wikipedia y portales gubernamentales.

El sistema utiliza procesamiento de lenguaje natural (NLP) basado en reglas y palabras clave para interpretar las consultas del usuario, limpiar el ruido lingüístico y ofrecer respuestas contextualizadas con un índice de confianza calculado matemáticamente.

### 2. ARQUITECTURA DEL SISTEMA

El software sigue el patrón de diseño **Capas (Layered Architecture)**, separando estrictamente la responsabilidad de la visualización, la lógica de búsqueda y la adquisición de datos.

#### 2.1 Diagrama de Componentes Lógicos

El sistema se divide en tres módulos principales que interactúan secuencialmente:

1. **Capa de Presentación (Frontend):** Gestionada por la clase `ChatbotGUI`. Es la interfaz visual con la que interactúa el usuario.
2. **Capa de Lógica de Negocio (Core):** Gestionada por `SearchEngine` y `QueryProcessor`. Es el cerebro que decide qué respuesta es la mejor.

3. **Capa de Datos (Data Access):** Gestionada por WebScraper. Es el "brazo robótico" que busca información en internet.



### 3. INGENIERÍA DETALLADA POR MÓDULOS

A continuación, se describe la lógica interna de cada clase implementada en el código.

#### 3.1. Clase WebScraper (Módulo de Adquisición)

Este módulo es responsable de la minería de datos. Su función es convertir el HTML desestructurado de la web en texto limpio y procesable.

- **Gestión de Sesiones:** Se utiliza `requests.Session()` en lugar de llamadas simples. Esto permite mantener cookies y, lo más importante, injectar un `User-Agent` de navegador real (`Mozilla/5.0`). Sin esto, servidores como Wikipedia bloquearían al bot por considerarlo un script malicioso.
- **Algoritmo de Limpieza (clean\_text):** El texto crudo de internet está "sucio". Esta clase implementa expresiones regulares (`re`) para:
  - Eliminar referencias académicas (ej: [12], [cita requerida]).
  - Normalizar espacios en blanco (convertir tabulaciones y dobles espacios en uno solo).

- **Filtro de Relevancia:** Descarta cualquier párrafo menor a 30 caracteres (ruido) o mayor a 300 (demasiado extenso para un chat), asegurando que el bot sea conciso.

### 3.2. Clase **QueryProcessor** (Módulo de NLP)

Antes de buscar una respuesta, el sistema debe "entender" la pregunta. Este módulo normaliza la entrada humana.

- **Eliminación de Stopwords:** El sistema cuenta con un set (set) de palabras vacías en español (*de, la, que, por, para*). Estas se eliminan de la consulta porque no aportan valor semántico, dejando solo los sustantivos y verbos clave.
- **Expansión Semántica (Sinónimos):** Implementa un diccionario de equivalencias.
  - *Ejemplo:* Si el usuario escribe "pantera", el sistema internamente busca también "jaguar". Si escribe "selva", busca "amazonía". Esto aumenta drásticamente la probabilidad de encontrar una respuesta correcta.
- **Tokenización y Categorización:** Divide la frase en palabras y detecta si la consulta se refiere a una *Especie* (ej: cóndor), una *Región* (ej: Altiplano) o un *Concepto* (ej: extinción).

### 3.3. Clase **SearchEngine** (Motor de Búsqueda)

Es el núcleo orquestador. Implementa un **Algoritmo de Búsqueda Híbrido** con puntuación ponderada. No busca coincidencias exactas simples, sino que "califica" cada oración de la base de datos para ver cuál se parece más a lo que el usuario pidió.

La Fórmula de Puntuación (Ranking Algorithm):

Para cada oración en la base de datos, se calcula un puntaje (\$\$\$) basado en:

$$\$ \$ \$ = (2 \times P_{\{\text{coincidencia}\}}) + (3 \times P_{\{\text{keyword}\}}) + (5 \times P_{\{\text{especie}\}}) + (3 \times P_{\{\text{region}\}}) \$ \$ \$$$

- **+2 puntos:** Si una palabra de la consulta aparece en la oración.
- **+3 puntos:** Si aparece una palabra clave general.
- **+5 puntos:** Si coincide el nombre de una especie (prioridad máxima).
- **+3 puntos:** Si coincide una región geográfica.

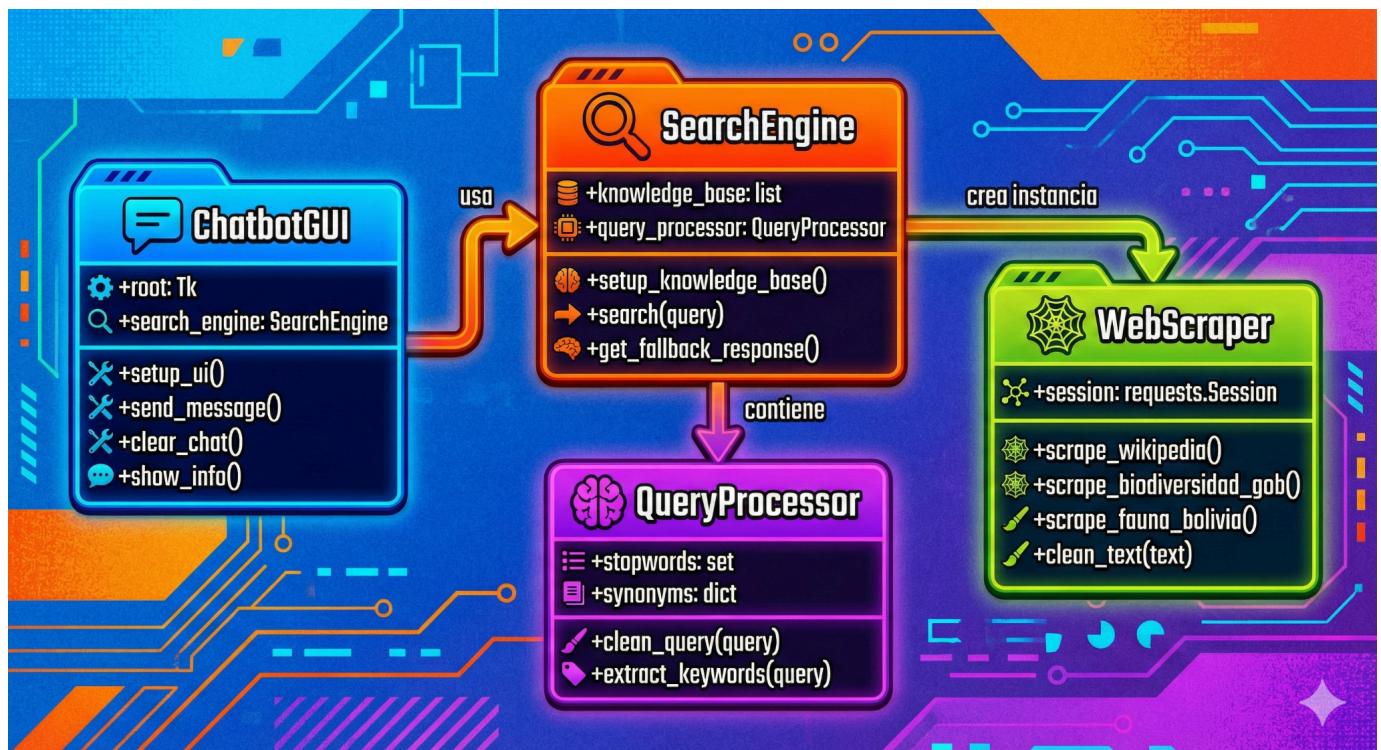
Cálculo de Confianza:

El sistema calcula un porcentaje de certeza (0.0 a 1.0). Si la mejor respuesta obtenida tiene una confianza menor a 0.2 (20%), el sistema descarta la respuesta y activa el protocolo de Fallback (respuesta por defecto), preguntando al usuario si puede reformular, evitando dar respuestas incorrectas o alucinaciones.

### 3.4. Clase ChatbotGUI (Interfaz de Usuario)

Desarrollada con **Tkinter**, la librería estándar de GUI de Python.

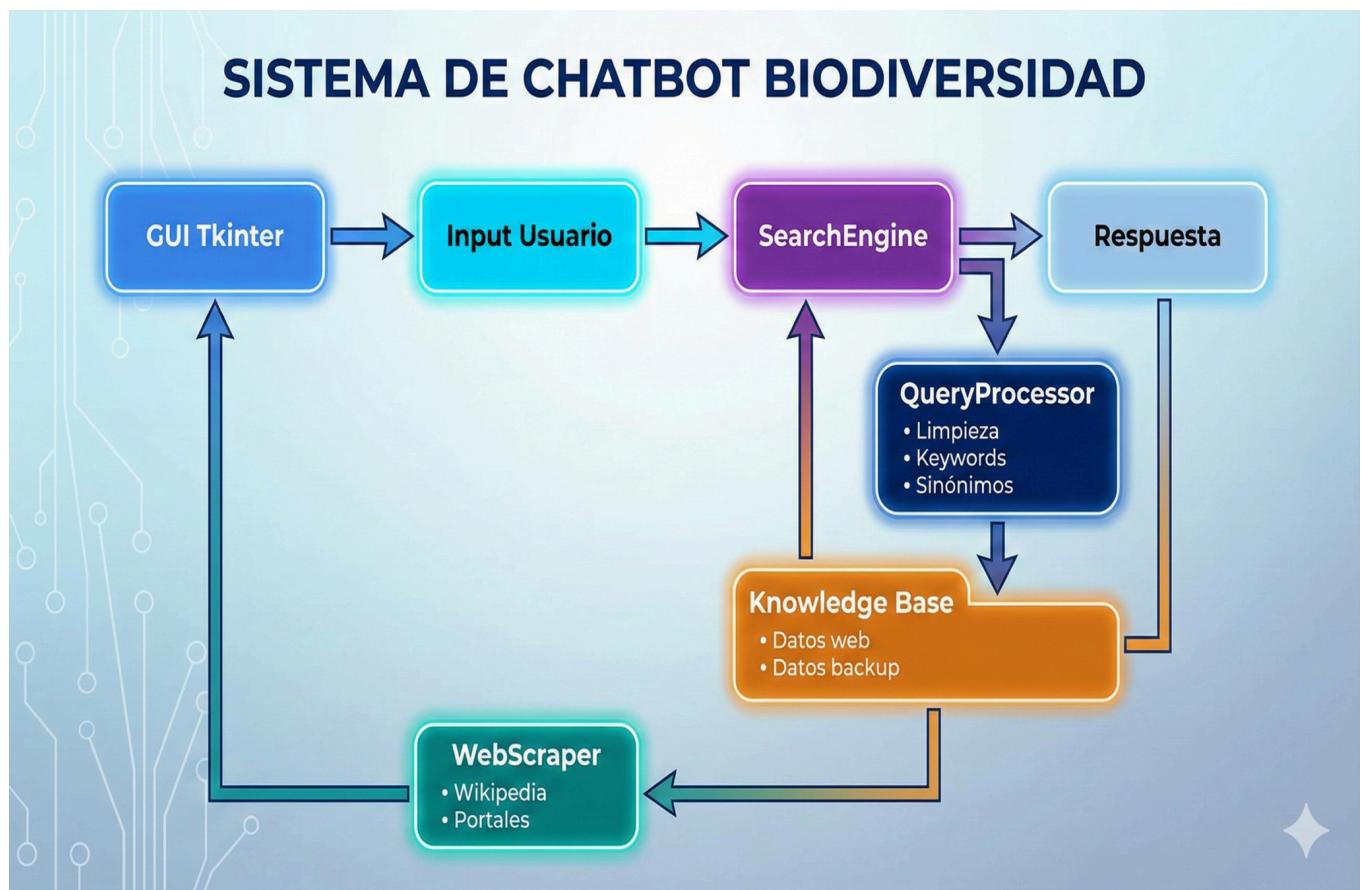
- **Diseño Asíncrono Simulado:** Aunque Tkinter es *single-threaded*, el método `root.after` o la estructura de eventos permite que la interfaz no se congele completamente mientras se procesan textos cortos.
- **Widgets:**
  - `ScrolledText`: Permite historiales de chat largos.
  - `Entry + Bind <Return>`: Permite enviar mensajes pulsando Enter, mejorando la experiencia de usuario (UX).



#### 4. FLUJO DE DATOS (DATA FLOW)

El ciclo de vida de una consulta (query) en el sistema es el siguiente:

1. **Input:** El usuario escribe: "*¿Qué come el oso jucumari?*"
2. **Preprocesamiento:**
  - Minúsculas: "*qué come el oso jucumari*"
  - Stopwords removidas: "*come oso jucumari*"
  - Sinónimos aplicados: "*come oso andino jucumari*"
3. **Búsqueda:** El motor recorre las ~50-100 oraciones cargadas (Wikipedia + Base Local).
4. **Ranking:** Encuentra la oración: "*El oso andino o jucumari se alimenta de bromelias en los bosques.*"
  - Coincide "oso" (+5 pts), "jucumari" (+5 pts). Puntaje alto.
5. **Output:** Se muestra la respuesta en pantalla y se indica la confianza (ej: 0.92).



## 5. GESTIÓN DE ERRORES Y CONTINGENCIAS

El código está diseñado bajo el principio de "**Fail-Safe**" (**A prueba de fallos**):

- **Fallo de Internet:** Si el `WebScraper` no puede conectar a Wikipedia (timeout o error 404), el bloque `try-except` captura el error, lo reporta en consola (`print`), pero **no detiene el programa**. En su lugar, carga silenciosamente los datos de respaldo (`backup_data`), garantizando que el usuario siempre reciba atención.
- **Consultas Vacías:** Se valida que el input no esté vacío antes de procesar para ahorrar recursos de CPU.

## 6. REQUISITOS TÉCNICOS Y DESPLIEGUE

Para la ejecución exitosa de este software en un entorno de producción o desarrollo, se requieren las siguientes especificaciones:

### Dependencias de Software

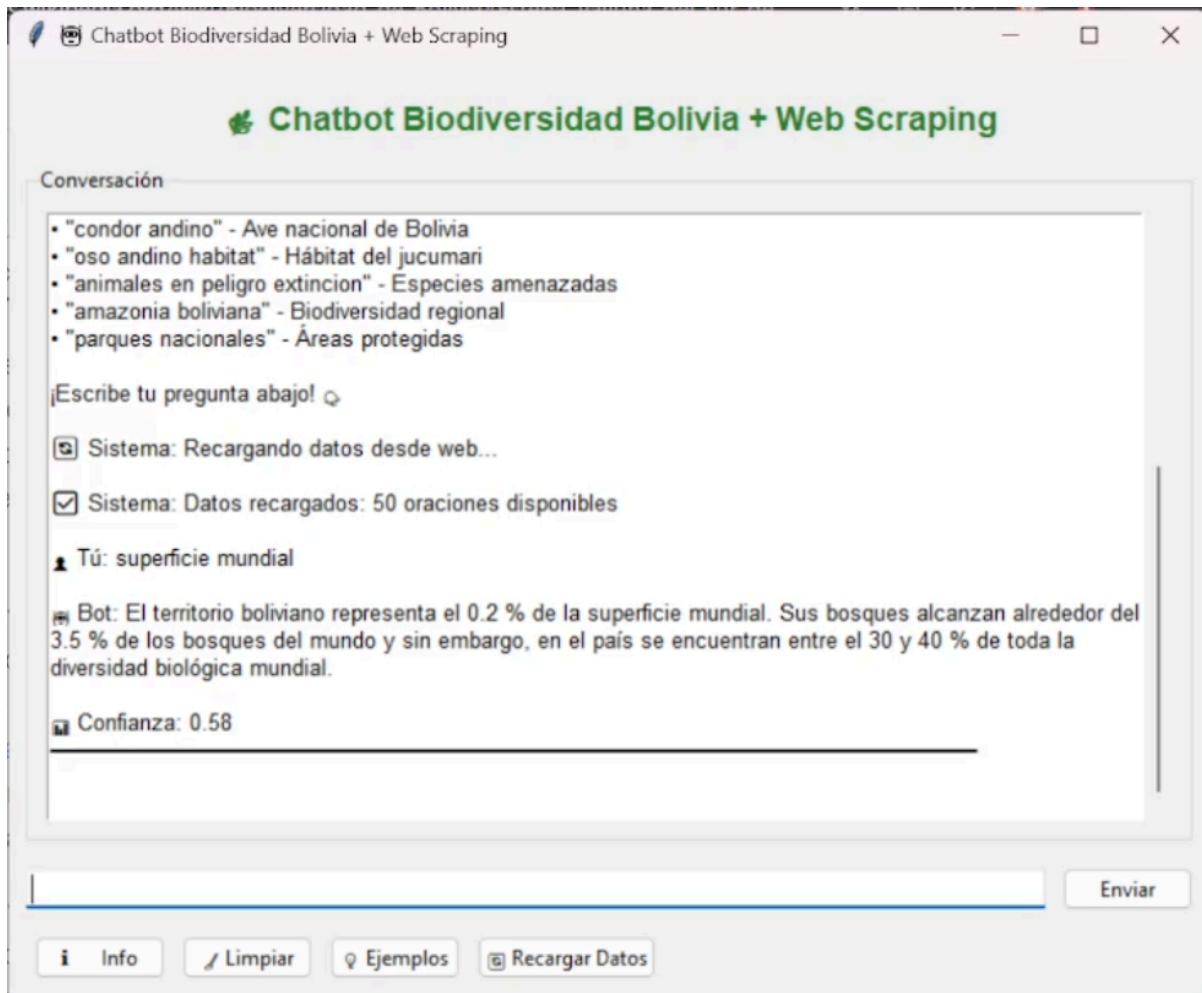
- **Lenguaje:** Python 3.8 o superior.
- **Librerías Externas:**
  - `requests`: Para peticiones HTTP.
  - `beautifulsoup4` (`bs4`): Para el parseo de HTML.
  - `lxml` (Opcional, pero recomendado para parseo rápido).

### Instalación

Bash

```
pip install requests beautifulsoup4
```

## LA CORRIDA DEL PROGRAMA



## URLS USADAS

```
https://es.wikipedia.org/wiki/Biodiversidad_de_Bolivia#:~:text=Distrito%20Chaque%C3%B1o:%20entre%20su%20fauna,realmente%20es%20el%20tibur%C3%A9n%20sarda.

https://es.wikipedia.org/wiki/Flora_de_Bolivia#:~:text=Estepa%20valluna,%20causa%20de,la%20tuna%20y%20el%20tumbo.

https://www.faunabolivia.com/
```

## 7. CONCLUSIÓN Y TRABAJO FUTURO

El código actual representa un prototipo funcional robusto de un sistema de recuperación de información basado en dominios específicos (Biodiversidad). La arquitectura modular permite una fácil escalabilidad.

### Mejoras Potenciales Identificadas:

1. **Persistencia:** Guardar la base de conocimientos "scrapeada" en un archivo local (JSON/SQLite) para evitar descargas repetitivas cada vez que se abre el programa.
2. **Modelos Vectoriales:** Reemplazar el conteo de palabras clave por *Word Embeddings* (Word2Vec o BERT) para entender el contexto semántico profundo (ej: entender que "alimentación" y "dieta" son lo mismo sin necesidad de un diccionario manual de sinónimos).
3. **Multimedia:** Habilitar la descarga y visualización de imágenes de las especies detectadas en el scraping.