Documentación Técnica:

API de Transcripción SurtiLab-GTranscriptor

1. Introducción

SurtiLab-GTranscriptor es un servicio de API RESTful diseñado para transcribir archivos de audio a texto. El servicio utiliza un modelo de machine learning avanzado (wav2vec2-xlsr-300m-guarani) especializado en la **transcripción del idioma Guaraní**.

Esta documentación detalla los recursos disponibles en la API, los métodos de integración, los requisitos de los datos y las mejores prácticas para asegurar una implementación robusta y escalable. La API está alojada en Hugging Face Spaces y es accesible públicamente.

Puntos Clave:

- **Funcionalidad Principal:** Conversión de Audio a Texto.
- **Idioma Soportado:** Guaraní.
- **Tecnología:** FastAPI, Transformers, PyTorch.
- **Modelo:** ivangtorre/wav2vec2-xlsr-300m-guarani.

2. Especificación de la API

2.1. Endpoint Principal

El único endpoint disponible es para solicitar una transcripción.

- **URL:** https://SurtiLab-GTranscriptor.hf.space/transcribir
- **Método HTTP:** POST
- **Descripción:** Envía un archivo de audio (por URL o en Base64) para ser procesado y devuelve su transcripción en texto.

2.2. Headers

Clave	Valor	Obligatorio	Descripción
Content-Type	application/json	Sí	Indica que el cuerpo de la petición es JSON.

2.3. Request Body

El cuerpo de la petición debe ser un objeto JSON que contenga **una de las siguientes dos claves**. Enviar ambas o ninguna resultará en un error.

Método 1: Vía URL Pública

Este es el método preferido si el audio ya está alojado en un servidor y es accesible públicamente.

- **Estructura JSON:**
```
{
  "url": "URL_PUBLICA_DEL_AUDIO"
}
```

- **url (string):** Una URL válida y accesible desde internet que apunte directamente al archivo de audio. El servidor descargará el contenido desde esta dirección.

Método 2: Vía Audio en Base64

Este método es ideal para transcribir audios locales o que no están expuestos en una URL pública.

- **Estructura JSON:**
```
{
  "base64_audio": "CADENA_BASE64_DEL_AUDIO"
}
```

- **base64_audio (string):** El contenido del archivo de audio codificado en formato Base64.

2.4. Respuestas del Servidor

Respuesta Exitosa (Código 200 OK)

Si la petición es válida y el audio se procesa correctamente, la API devolverá un objeto JSON con la transcripción.

- **Estructura JSON:**
  ```
  {
    "transcripcion": "texto de la transcripción en minúsculas"
  }
  ```

- **transcripcion (string):** El texto resultante del proceso de transcripción. El texto se devuelve siempre en minúsculas.

Respuestas de Error (Código 400 Bad Request)

Si hay un problema con la petición, la API devolverá un error con un mensaje descriptivo.

- **Estructura JSON:**
  ```
  {
    "detail": "Mensaje de error descriptivo"
  }
  ```

- **Posibles Mensajes de Error:**
  - "Falta 'url' o 'base64_audio'.": No se proporcionó ninguna de las dos claves requeridas.
  - "Error al descargar el archivo.": La URL proporcionada no es válida, no es accesible o no devolvió un código 200.
  - "Error al convertir audio: {detalle}": El formato del audio es corrupto o no pudo ser procesado por el servidor.

3. Requisitos del Audio

Aunque la API es flexible, es importante conocer cómo se procesa el audio para optimizar los resultados.

- **Formatos Soportados:** Gracias a la librería pydub en el backend, la API puede procesar una amplia variedad de formatos de audio comunes, incluyendo:
  - WAV
  - MP3
  - OGG
  - FLAC
  - AAC
  - M4A
  - ...entre otros.
- **Procesamiento Interno: Independientemente del formato de entrada**, el servidor convierte automáticamente todo el audio a las siguientes especificaciones antes de pasarlo al modelo:
  - **Formato:** WAV
  - **Frecuencia de Muestreo:** 16000 Hz (16kHz)
  - **Canales:** 1 (Mono)
- **Recomendación:** Para obtener la máxima velocidad y evitar posibles errores de conversión, se recomienda pre-procesar el audio a **WAV, 16kHz, Mono** antes de enviarlo a la API, aunque no es un requisito estricto.

4. Ejemplos de Integración

A continuación, se muestran ejemplos de código completos para integrarse con la API.

4.1. Integración en JavaScript (Navegador)

a) Usando una URL de Audio

Este es el caso de uso del código que proporcionaste, adaptado al estándar fetch.

```
// URL del endpoint de la API
const API_ENDPOINT =
'[https://SurtiLab-GTranscriptor.hf.space/transcribir](https://SurtiLab-GTranscriptor.hf.space/transcribir)';

// URL pública del archivo de audio que se desea transcribir
const audioUrl =
'[https://ejemplo.com/mi-audio-en-guarani.mp3](https://ejemplo.com/mi-audio-en-guarani.mp3)';

async function transcribirDesdeUrl(url) {
  console.log('Iniciando transcripción para la URL:', url);

  const options = {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      "url": url
    })
  };

  try {
    const response = await fetch(API_ENDPOINT, options);

    if (!response.ok) {
      // Si la respuesta no es 2xx, obtenemos el error del cuerpo
      const errorData = await response.json();
      throw new Error(`Error del servidor: ${response.status} - ${errorData.detail}`);
    }

    const data = await response.json();
    const transcripcion = data.transcripcion;

    console.log('Transcripción recibida:', transcripcion);
```

```
    return transcripcion;

  } catch (error) {
    console.error('No se pudo procesar el audio. El error fue:', error);
    // Aquí puedes manejar el error, por ejemplo, mostrando un mensaje al usuario.
    return null;
  }
}

// Llamar a la función para iniciar el proceso
transcribirDesdeUrl(audioUrl);
```

b) Usando un Archivo Local (Base64)

Este ejemplo muestra cómo leer un archivo desde un <input type="file">, convertirlo a Base64 y enviarlo.

```
<!-- HTML necesario para la carga de archivos -->
<label for="audio_file">Selecciona un archivo de audio:</label>
<input type="file" id="audio_file" accept="audio/*">
<button id="transcribirBtn">Transcribir</button>
<p><strong>Resultado:</strong> <span id="resultado"></span></p>

<script>
  const API_ENDPOINT =
'[https://SurtiLab-GTranscriptor.hf.space/transcribir](https://SurtiLab-GTranscriptor.hf.space/transcribir)';

  // Función para convertir un archivo a Base64
  function fileToBase64(file) {
    return new Promise((resolve, reject) => {
      const reader = new FileReader();
      reader.readAsDataURL(file);
      reader.onload = () => {
        // La cadena Base64 incluye un prefijo (ej: "data:audio/wav;base64,"), lo eliminamos.
        const base64String = reader.result.split(',')[1];
        resolve(base64String);
      };
      reader.onerror = error => reject(error);
    });
  }

  async function transcribirDesdeBase64(base64Audio) {
    document.getElementById('resultado').innerText = 'Procesando...';
```

```
    const options = {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ "base64_audio": base64Audio })
    };

    try {
      const response = await fetch(API_ENDPOINT, options);
      if (!response.ok) {
        const errorData = await response.json();
        throw new Error(`Error: ${errorData.detail}`);
      }
      const data = await response.json();
      document.getElementById('resultado').innerText = data.transcripcion;
    } catch (error) {
      console.error('Error en la transcripción:', error);
      document.getElementById('resultado').innerText = `Error: ${error.message}`;
    }
  }

  // Event Listener para el botón
  document.getElementById('transcribirBtn').addEventListener('click', async () => {
    const fileInput = document.getElementById('audio_file');
    if (fileInput.files.length === 0) {
      alert('Por favor, selecciona un archivo primero.');
      return;
    }
    const file = fileInput.files[0];
    try {
      const base64Audio = await fileToBase64(file);
      await transcribirDesdeBase64(base64Audio);
    } catch (error) {
      console.error("Error al convertir a Base64:", error);
      document.getElementById('resultado').innerText = "Error al leer el archivo.";
    }
  });
</script>
```

4.2. Integración en Python (Backend)

Este ejemplo es útil para integraciones de servidor a servidor.

```
import requests
```

```
import base64

API_ENDPOINT =
"[https://SurtiLab-GTranscriptor.hf.space/transcribir](https://SurtiLab-GTranscriptor.hf.space/tran
scribir)"
HEADERS = {"Content-Type": "application/json"}

def transcribir_desde_url(audio_url: str):
    """Solicita la transcripción de un audio desde una URL pública."""
    payload = {"url": audio_url}
    try:
        response = requests.post(API_ENDPOINT, json=payload, headers=HEADERS)
        response.raise_for_status()  # Lanza una excepción para códigos de error HTTP
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Error en la petición: {e}")
        # A menudo la respuesta contiene más detalles del error
        if e.response:
            print(f"Detalle del servidor: {e.response.text}")
        return None

def transcribir_desde_archivo_local(file_path: str):
    """Lee un archivo local, lo codifica en Base64 y solicita la transcripción."""
    try:
        with open(file_path, "rb") as audio_file:
            encoded_string = base64.b64encode(audio_file.read()).decode('utf-8')

        payload = {"base64_audio": encoded_string}
        response = requests.post(API_ENDPOINT, json=payload, headers=HEADERS)
        response.raise_for_status()
        return response.json()
    except FileNotFoundError:
        print(f"Error: El archivo no se encontró en la ruta: {file_path}")
        return None
    except requests.exceptions.RequestException as e:
        print(f"Error en la petición: {e}")
        if e.response:
            print(f"Detalle del servidor: {e.response.text}")
        return None

# --- EJEMPLOS DE USO ---

# 1. Usando URL
url_guarani =
```

```
"[https://f003.backblazeb2.com/file/hf-guarani-dataset/guarani_test_0.mp3](https://f003.backblaz
eb2.com/file/hf-guarani-dataset/guarani_test_0.mp3)" # URL de ejemplo
resultado_url = transcribir_desde_url(url_guarani)
if resultado_url:
    print(f"Transcripción (URL): {resultado_url['transcripcion']}")

# 2. Usando un archivo local (asegúrate de tener un archivo 'audio.mp3' en la misma carpeta)
archivo_local = "audio.mp3"
resultado_local = transcribir_desde_archivo_local(archivo_local)
if resultado_local:
    print(f"Transcripción (Local): {resultado_local['transcripcion']}")
```

5. Consideraciones para un Entorno Escalable

- **Manejo de Tiempos de Espera (Timeouts):** La transcripción de audios largos puede tomar tiempo. Configura timeouts generosos en tu cliente HTTP para evitar que la conexión se cierre prematuramente.
- **Gestión Asíncrona:** En aplicaciones de cara al usuario, no bloquees la interfaz mientras esperas la respuesta. Muestra un indicador de carga (spinner) y actualiza la UI cuando la respuesta llegue.
- **Manejo de Errores Robusto:** Implementa una lógica de reintentos (con *exponential backoff*) para errores de red o errores temporales del servidor (códigos 5xx). Para errores de cliente (4xx), informa al usuario sobre el problema (ej: "El formato del audio no es válido").
- **Límites de Recursos:** El servicio se ejecuta en un *Space* de Hugging Face, que tiene recursos de cómputo y memoria limitados. Evita enviar un número masivo de peticiones simultáneas. Si necesitas alto rendimiento, considera implementar una cola de trabajos (ej: RabbitMQ, Celery) para procesar las solicitudes de manera ordenada.
- **Seguridad:** La API actual es pública y no requiere autenticación. Para un entorno de producción o para proteger el servicio contra abusos, el siguiente paso sería implementar un sistema de claves de API (API Keys). Esto se podría añadir en el backend de FastAPI verificando una cabecera como X-API-Key.