



UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIRÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIRÍA EN TELECOMUNICACIONES

TRABAJO DE INTEGRACIÓN CURRICULAR

TEMA:

“PROPUESTA PARA LA IMPLEMENTACIÓN DE UN ASISTENTE
CONVERSACIONAL CON INTELIGENCIA ARTIFICIAL INTEGRADO AL ISSABEL
CALL CENTER PARA LA RESOLUCIÓN AUTOMÁTICA DE CONSULTAS BÁSICAS”

Trabajo de titulación previo a la obtención del título en Telecomunicaciones

Línea de investigación: Producción industrial y tecnología sostenible

AUTOR:

Luis Roberto Chuquin Guaman

DIRECTOR:

MSc. Jaime Roberto Michilena Calderon

Ibarra – Ecuador 2026



UNIVERSIDAD TÉCNICA DEL NORTE

DIRECCIÓN DE BIBLIOTECA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD

TÉCNICA DEL NORTE

IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO	
APELLIDOS Y NOMBRES:	CHUQUIN GUAMAN LUIS ROBERTO

DATOS DE LA OBRA	
TÍTULO:	“PROPUESTA PARA LA IMPLEMENTACIÓN DE UN ASISTENTE CONVERSACIONAL CON INTELIGENCIA ARTIFICIAL INTEGRADO AL ISSABEL CALL CENTER PARA LA RESOLUCIÓN AUTOMÁTICA DE CONSULTAS BÁSICAS”
AUTOR (ES):	CHUQUIN GUAMAN LUIS ROBERTO
FECHA: DD/MM/AAAA	23/10/2026
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	☒ PREGRADO ☐ POSGRADO

TITULO POR EL QUE OPTA:	INGENIERO EN TELECOMUNICACIONES
DIRECTOR /ASESOR:	MSC. JAIME ROBERTO CHICHILENA CALDERON MSC. CARLOS ALBERTO VÁSQUEZ AYALA

CONSTANCIAS

El autor (es) manifiesta (n) que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto la obra es original y que es (son) el (los) titular (es) de los derechos patrimoniales, por lo que asume (n) la responsabilidad sobre el contenido de la misma y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 23 días del mes de octubre de 2026

EL AUTOR:



.....

Nombre: Chuquin Guaman Luis Roberto.

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE
INTEGRACIÓN CURRICULAR

Ibarra,

MSc. JAIME ROBERTO MICHILENA CALDERÓN

DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

CERTIFICA:

Haber revisado el presente informe final del trabajo de Integración Curricular, el mismo que se ajusta a las normas vigentes de la Universidad Técnica del Norte; en consecuencia, autorizo su presentación para los fines legales pertinentes.



(f)

Ing. Jaime Roberto Michilena Calderón, MsC.

C.C.: 1002198438

DEDICATORIA

El presente trabajo va dedicado en primer lugar a mi madre por ser la mujer fuerte y valiente quien me enseñó la perseverancia y el trabajo duro, de la misma manera a mis hermanos Oswaldo, Luis, Elsa, Ivan, Victoria, Carlos y Adela quienes me apoyaron en cada una de las etapas, así también a mis amigos y compañeros quienes fueron un apoyo en todo el camino de la universidad.

AGRADECIMIENTO

A todas las personas, amigos, hermanos, sobrinos quienes me vieron empezar un largo camino y fueron quienes no dejaron que me rinda en el transcurso de mi carrera profesional, en especial a mi madre Blanquita que sin su apoyo no podría haber conseguido este título, también agradezco a mis hermanos por sus palabras de apoyo y por haber puesto un granito de arena, también agradezco a mis docentes quienes fueron parte de este largo proceso que sin sus enseñanzas este logro no sería el mismo, gracias a todos quienes me formaron como persona y como profesional.

RESUMEN

El presente trabajo tuvo como finalidad diseñar, implementar y evaluar un asistente conversacional de voz integrado a una central telefónica IP basada en Issabel/Asterisk, con el propósito de automatizar la atención de consultas sencillas mediante interacción natural entre el usuario y el sistema. Para ello, se definió una arquitectura distribuida compuesta por una PBX en Issabel, un script AGI desarrollado en PHP para el control del flujo de llamadas y tres microservicios implementados en Python, orientados al reconocimiento de voz, la generación de respuestas en lenguaje natural y la síntesis de voz.

Durante la fase de diseño se establecieron los requerimientos funcionales y no funcionales de la solución, así como los recursos de hardware, las dependencias de software, los puertos de comunicación y la distribución funcional de los lenguajes empleados. A partir de este análisis se seleccionaron tecnologías de software libre como Issabel, Asterisk, Python, Vosk, Gemini y Piper, debido a su flexibilidad, compatibilidad, facilidad de integración y reducción de costos. Esta etapa permitió definir una base técnica sólida para la implementación posterior, evitando decisiones improvisadas y asegurando coherencia entre los componentes del sistema.

En la fase de implementación se configuraron las extensiones SIP, el entorno IVR y la comunicación interna en Issabel, validando inicialmente el correcto funcionamiento de las llamadas entre terminales softphone. Posteriormente, se desarrolló el script AGI encargado de contestar la llamada, grabar la voz del usuario, enviar el audio al servicio de transcripción, remitir el texto reconocido al módulo de generación de respuestas con Gemini y solicitar al servicio TTS la conversión de la respuesta textual en audio reproducible por Asterisk. Paralelamente, se configuraron los servicios Python: uno basado en Vosk para la transcripción de voz a texto, otro apoyado en Gemini para la interpretación de consultas y

generación de respuestas, y un tercero basado en Piper para la síntesis de voz en formato compatible con telefonía.

Una vez integrados todos los componentes, se realizaron pruebas funcionales y de desempeño para verificar la operación del sistema en un entorno real de llamada. Como criterio de evaluación, se formularon 10 preguntas sencillas al asistente, además de una instrucción de salida para finalizar la conversación. Estas pruebas permitieron comprobar la comunicación efectiva entre módulos, la continuidad del flujo conversacional y la capacidad del sistema para responder automáticamente a consultas básicas mediante voz. Los resultados evidenciaron que la arquitectura implementada logró atender la totalidad de las consultas sin intervención humana, alcanzando una tasa de automatización bruta del 100 %, una precisión útil de respuesta del 90 % y un reconocimiento final exitoso del 100 % de las preguntas planteadas. Asimismo, se estimó un tiempo promedio de procesamiento de 24.2 segundos y un tiempo total de respuesta percibida de 30.1 segundos por interacción.

El trabajo permitió demostrar la viabilidad de integrar servicios de inteligencia artificial con una plataforma PBX de software libre para construir un asistente conversacional telefónico funcional. Aunque se identificaron oportunidades de mejora, especialmente en la reducción de la latencia y en la optimización de respuestas dependientes del contexto temporal, la solución desarrollada constituye una base sólida para futuras ampliaciones, como la incorporación de bases de datos, el tratamiento de consultas más complejas o la adaptación del sistema a escenarios reales de atención automatizada.

Palabras clave: Asistente conversacional, Telefonía IP, Issabel, Asterisk, Reconocimiento de voz, Síntesis de voz, Inteligencia artificial, Procesamiento de lenguaje natural, Automatización de consultas, Interacción por voz

ABSTRACT

This work aimed to design, implement, and evaluate a voice-based conversational assistant integrated into an IP telephony system based on Issabel/Asterisk, with the purpose of automating the handling of simple inquiries through natural interaction between the user and the system. To achieve this, a distributed architecture was defined, consisting of an Issabel-based PBX, an AGI script developed in PHP to control the call flow, and three Python-based microservices focused on speech recognition, natural language response generation, and speech synthesis.

During the design phase, the functional and non-functional requirements of the solution were established, along with the hardware resources, software dependencies, communication ports, and the functional distribution of the programming languages used. Based on this analysis, open-source technologies such as Issabel, Asterisk, Python, Vosk, Gemini, and Piper were selected due to their flexibility, compatibility, ease of integration, and cost reduction. This stage made it possible to define a solid technical foundation for the subsequent implementation, avoiding improvised decisions and ensuring consistency among the system components.

In the implementation phase, SIP extensions, the IVR environment, and internal communication in Issabel were configured, initially validating the correct operation of calls between softphone terminals. Subsequently, the AGI script was developed to answer the call, record the user's voice, send the audio to the transcription service, forward the recognized text to the Gemini-based response generation module, and request from the TTS service the conversion of the textual response into audio playable by Asterisk. At the same time, the Python services were configured: one based on Vosk for speech-to-text transcription, another

supported by Gemini for query interpretation and response generation, and a third based on Piper for speech synthesis in a telephony-compatible format.

Once all components were integrated, functional and performance tests were conducted to verify system operation in a real call environment. As an evaluation criterion, 10 simple questions were asked to the assistant, in addition to an exit command used to end the conversation. These tests made it possible to verify effective communication between modules, continuity of the conversational flow, and the system's ability to respond automatically to basic voice-based inquiries. The results showed that the implemented architecture was able to handle all queries without human intervention, achieving a gross automation rate of 100%, a useful response accuracy of 90%, and a final successful recognition rate of 100% for the questions asked. Likewise, an average processing time of 24.2 seconds and a total perceived response time of 30.1 seconds per interaction were estimated.

This work demonstrated the feasibility of integrating artificial intelligence services with an open-source PBX platform in order to build a functional telephone-based conversational assistant. Although opportunities for improvement were identified, especially in reducing latency and optimizing responses dependent on temporal context, the developed solution constitutes a solid foundation for future extensions, such as database integration, the handling of more complex queries, or the adaptation of the system to real-world automated service scenarios.

Keywords: Conversational assistant, IP telephony, Issabel, Asterisk, speech recognition, speech synthesis, artificial intelligence, natural language processing, query automation, voice interaction.

ÍNDICE DE CONTENIDOS

Capítulo I: Antecedentes	18
1.1. Tema.....	18
1.2. Problema	18
1.3. Objetivos	19
<i>1.3.1. Objetivo General</i>	<i>19</i>
<i>1.3.2. Objetivos Específicos.....</i>	<i>19</i>
1.4. Alcance	20
1.5. Justificación	23
Capítulo II: Marco teórico y fundamentos de la investigación	25
2.1. Inteligencia Artificial y sus Fundamentos	25
2.2. Procesamiento de Lenguaje Natural (PLN)	26
<i>2.2.1. Modelos de Clasificación de Intenciones</i>	<i>27</i>
2.3. Grandes Modelos de Lenguaje (LLM).....	28
<i>2.3.1. IA Generativa y Asistentes Conversacionales</i>	<i>29</i>
2.4. Reconocimiento Automático de Voz (ASR) y Síntesis de Voz (TTS)	30
2.5. Arquitectura del Issabel Call Center	30
2.6. Telefonía IP y Protocolo VoIP	31
<i>2.6.1. Asterisk y la Interfaz AGI</i>	<i>32</i>
2.7. Asistentes Conversacionales en Centros de Atención	33
2.8. Transformación Digital en el Sector de Atención al Cliente en Ecuador ..	34
2.9. Arquitectura del Sistema Propuesto	36

Capítulo III: Levantamiento de requerimientos y diseño	38
3.1 Metodología de Desarrollo	38
3.2 Requerimientos del sistema.....	39
3.3 Requerimientos de hardware.....	41
3.4 Requerimientos de software.....	43
3.5 Elección del servidor PBX y del motor de telefonía.....	46
3.6 Elección de los lenguajes de programación	46
3.7 Elección del modelo de transcripción de voz.....	48
3.8 Elección del modelo generativo para la respuesta	48
3.9 Elección del sintetizador de voz	49
3.10 Elección del modelo de despliegue y del servidor de servicios.....	49
3.11 Elección del cliente SIP para pruebas.....	50
3.12 Síntesis de lo realizado en la fase de diseño	50
Capítulo IV: Evaluación del prototipo.....	52
4.1 Entorno de implementación	52
4.2 Configuración inicial de Issabel.....	54
4.3 Creación de extensiones SIP y validación de llamadas internas.....	55
4.4 Configuración del IVR base en Issabel	56
4.5 Desarrollo del script AGI en PHP	57
4.6 Implementación del servicio de transcripción con Vosk.....	59
4.7 Implementación del servicio de generación de respuestas con Gemini	61

4.8 Implementación del servicio de síntesis de voz con Piper	62
4.9 Integración de la arquitectura completa.....	63
4.10 Ajustes y resolución de incidencias	65
4.11 Arquitectura final implementada.....	66
4.12 Evaluación del asistente conversacional implementado sobre Issabel.....	67
4.12.1 Metodología de evaluación	67
4.12.2 Resultados obtenidos.....	69
4.12.3 Análisis de indicadores	74
4.12.4 Conclusión de la evaluación.....	75
4.13 Resultados de validación	75
5. Conclusiones	76
6. Recomendaciones	77
7. REFERENCIAS.....	80
8. Anexos	83
Anexo 1: Código fuente de programación para transcripción de voz a texto	83
Anexo 2: Código fuente del servicio de generación de respuestas con Gemini.....	86
Anexo 3: Código fuente del servicio de síntesis de voz.....	90
Anexo 4: Archivo de configuración del entorno	94
Anexo 5: Código fuente del script AGI de integración telefónica	95
Anexo 6: Pruebas realizadas a el sistema usando 10 preguntas sencillas	101

ÍNDICE DE FIGURAS

Figura 1 <i>Diagrama de desarrollo</i>	21
Figura 2 <i>Call center de la empresa Bangara-Quito</i>	36
Figura 3 <i>Arquitectura del sistema propuesto</i>	37
Figura 4 <i>Esquema de diseño final</i>	51
Figura 5 <i>Comprobación y configuraciones iniciales en servidor Isabell</i>	54
Figura 6 <i>Creación de extensiones para realizar pruebas además de un IVR</i>	55
Figura 7 <i>Prueba de marcación entre extensiones</i>	55
Figura 8 <i>Configuración de IVR</i>	56
Figura 9 <i>Extracto del código de programación del archivo AGI</i>	58
Figura 10 <i>Extracto de código de programación para la transcripción</i>	60
Figura 11 <i>Extracto de código de programación para Gemini</i>	61
Figura 12 <i>Extracto de código de programación del archivo para TTS</i>	63
Figura 13 <i>Implementación del proyecto en conjunto</i>	64
Figura 14 <i>Comprobación de funcionamiento realizando llamada al número 800</i>	65
Figura 15 <i>Pregunta 1-saludo</i>	102
Figura 16 <i>Pregunta 2</i>	102
Figura 17 <i>Pregunta 3</i>	102
Figura 18 <i>Pregunta 4</i>	103
Figura 19 <i>Pregunta 5</i>	103
Figura 20 <i>Pregunta 6</i>	103
Figura 21 <i>Pregunta 7</i>	103
Figura 22 <i>Pregunta 8</i>	103
Figura 23 <i>Pregunta 9</i>	104
Figura 24 <i>Pregunta 10</i>	104

Figura 25 Palabra clave de cierre de cesión (Muchas gracias)	104
---	-----

ÍNDICE DE TABLAS

Tabla 1	<i>Requerimientos funcionales y no funcionales del sistema propuesto</i>	<i>39</i>
Tabla 2	<i>Requerimientos de hardware considerados en la solución</i>	<i>41</i>
Tabla 3	<i>Software seleccionado para la arquitectura conversacional</i>	<i>43</i>
Tabla 4	<i>Lenguajes y marcos de desarrollo seleccionados.....</i>	<i>47</i>
Tabla 5	<i>Componentes y recursos empleados en la implementación.....</i>	<i>53</i>
Tabla 6	<i>Flujo funcional de la arquitectura conversacional implementada.....</i>	<i>66</i>
Tabla 7	<i>Preguntas básicas empleadas para la evaluación funcional del asistente.....</i>	<i>68</i>
Tabla 8	<i>Resultados detallados por pregunta, transcripción y tiempos estimados.....</i>	<i>70</i>
Tabla 9	<i>Indicadores consolidados de precisión, automatización y eficiencia.....</i>	<i>72</i>

Capítulo I: Antecedentes

1.1. Tema

PROPUESTA PARA LA IMPLEMENTACIÓN DE UN ASISTENTE CONVERSACIONAL CON INTELIGENCIA ARTIFICIAL INTEGRADO AL ISSABEL CALL CENTER PARA LA RESOLUCIÓN AUTOMÁTICA DE CONSULTAS BÁSICAS

1.2. Problema

En la actualidad, muchas organizaciones que utilizan plataformas de comunicación basadas en Issabel Call Center enfrentan dificultades para gestionar de manera eficiente el alto volumen de consultas básicas que realizan los usuarios. Estas solicitudes suelen estar relacionadas con información general, estados de trámites, horarios de atención, requisitos de servicios, soporte inicial o preguntas frecuentes. Sin embargo, debido a la limitada cantidad de agentes disponibles, estos requerimientos generan tiempos de espera prolongados, saturación de las líneas telefónicas y reducción de la calidad del servicio ofrecido [1].

Issabel Call Center, aunque es una solución robusta de telefonía IP y gestión de llamadas, depende principalmente de la intervención humana para la atención inicial. Esto implica que los agentes deben destinar una parte considerable de su tiempo a resolver consultas repetitivas y de bajo nivel, lo que afecta su disponibilidad para atender casos complejos o prioritarios. Además, la ausencia de un sistema automatizado que clasifique intenciones o responda preguntas frecuentes provoca que la atención sea poco escalable y que existan variaciones en la calidad del servicio según la carga operativa del momento [2].

Por ello se evidencia la necesidad de incorporar tecnologías de inteligencia artificial y procesamiento de lenguaje natural (PLN) que permitan automatizar la atención de consultas simples antes de ser derivadas a un agente humano. Un asistente conversacional integrado al Issabel Call Center podría desempeñar esta función, gestionando automáticamente preguntas básicas, reduciendo la carga de los agentes y mejorando la experiencia del usuario mediante tiempos de respuesta más rápidos y consistentes [3].

La falta de automatización ocasiona que los recursos humanos se utilicen de manera ineficiente, que los usuarios experimenten demoras innecesarias y que el sistema de atención telefónica no aproveche los avances actuales en inteligencia artificial. Por ello, surge la necesidad de proponer una solución tecnológica que permita integrar un asistente conversacional basado en IA al Issabel Call Center, con el fin de mejorar la capacidad de respuesta, optimizar los procesos operativos y ofrecer un servicio más ágil y eficiente, por lo que la investigación quedara para conocimiento público sin implementar en un sitio específico, quedara como una propuesta [4].

1.3. Objetivos

1.3.1. Objetivo General

Proponer la implementación de un asistente conversacional basado en inteligencia artificial integrado al Issabel Call Center, con la finalidad de automatizar la atención de consultas básicas.

1.3.2. Objetivos Específicos

- Analizar los fundamentos teóricos relacionados con el procesamiento de lenguaje natural, la inteligencia artificial aplicada a sistemas conversacionales y la arquitectura del Issabel Call Center.
- Diseñar la estructura funcional del asistente conversacional considerando los flujos de interacción, el modelo de clasificación de intenciones y la integración mediante la interfaz AGI de Issabel.
- Desarrollar un prototipo básico del asistente conversacional que permita reconocer e interpretar solicitudes frecuentes de los usuarios, respondiendo automáticamente a consultas simples.
- Evaluar el funcionamiento del asistente conversacional mediante indicadores simples de precisión y tiempo de respuesta.

1.4. Alcance

El presente estudio se desarrollará en cuatro fases claramente definidas que permitirán establecer, diseñar y evaluar la propuesta de un asistente conversacional con inteligencia artificial integrado al Issabel Call Center. Cada fase delimita las actividades y resultados esperados dentro del marco de la investigación.

Fase 1: Investigación y análisis

En esta fase se realizará la recopilación, revisión y análisis de la información necesaria para comprender el funcionamiento del procesamiento de lenguaje natural, los sistemas conversacionales basados en inteligencia artificial y la arquitectura tecnológica de Issabel Call Center. Se analizarán estudios previos, documentación técnica, modelos de diálogo, mecanismos de clasificación de intenciones y el funcionamiento de la interfaz AGI. Además, se identificarán

las principales limitaciones actuales del sistema de atención telefónica y las oportunidades de automatización mediante IA.

Fase 2: Levantamiento de requerimientos

Durante esta fase se definirán los requerimientos funcionales y no funcionales del asistente conversacional. Esto incluye la identificación de las consultas básicas más frecuentes, las condiciones de integración con Issabel, los parámetros operativos del modelo de IA, las restricciones tecnológicas, los criterios de respuesta y los recursos necesarios para la implementación del prototipo. Los requerimientos se establecerán considerando la viabilidad técnica y operativa del sistema dentro del entorno Issabel.

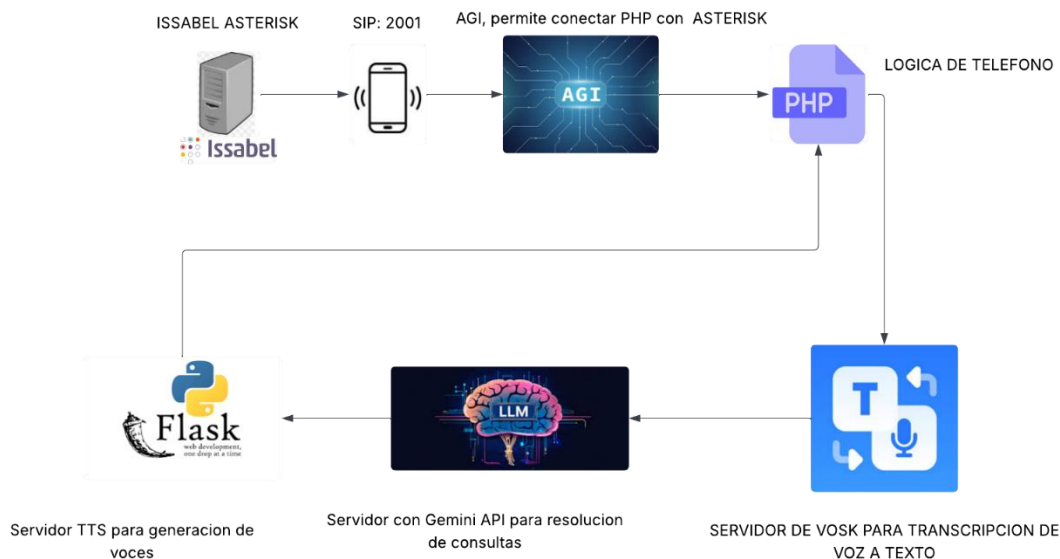
Fase 3: Diseño de la solución

En esta fase se elaborará la propuesta de arquitectura del asistente conversacional, como se indica en la figura 1, detallando los flujos de interacción, el modelo de clasificación de intenciones, la estructura del diálogo y el mecanismo de comunicación con Issabel mediante AGI. Se definirá el diseño del prototipo, especificando los módulos, componentes, funciones, diagramas de interacción y lógica de automatización. El diseño servirá como base técnica para la construcción del prototipo funcional.

El dispositivo del usuario final actúa únicamente como un medio de interacción, capturando la entrada de voz y reproduciendo las respuestas generadas, mientras que el procesamiento de intenciones, la generación de respuestas y la integración con el sistema Issabel se realizan en servicios backend. Esto evita dependencias del hardware del usuario y permite una implementación homogénea dentro del entorno del call center.

Figura 1

Diagrama de desarrollo



Fase 4: Evaluación del prototipo

Finalmente, la investigación contemplará la evaluación del prototipo implementado, mediante indicadores sencillos como precisión en la clasificación de intenciones, coherencia de las respuestas y tiempo de respuesta frente a consultas básicas. La evaluación permitirá determinar el grado de efectividad del asistente conversacional, identificar posibles mejoras y establecer su potencial de implementación real dentro del Issabel Call Center.

La evaluación se realizará mediante una metodología experimental, complementada con observaciones. Dicha metodología permitirá analizar el comportamiento del prototipo bajo condiciones controladas, simulando escenarios reales de atención al cliente dentro del call center.

Para ello, se definirá un conjunto de casos de prueba basados en consultas frecuentes, tales como solicitudes de información básica, soporte técnico y redireccionamiento de llamadas.

Estas pruebas se ejecutarán de forma repetitiva con el fin de obtener datos confiables y comparables.

1.5. Justificación

En Ecuador, los centros de atención telefónica constituyen un componente fundamental para la comunicación entre instituciones públicas, empresas privadas y usuarios. Sin embargo, gran parte de estos sistemas continúa operando bajo modelos tradicionales, donde la atención inicial depende exclusivamente de agentes humanos. Esta situación genera limitaciones operativas, especialmente en entornos donde el volumen de llamadas es elevado y se presentan numerosas consultas repetitivas o de baja complejidad, lo cual provoca tiempos de espera prolongados, saturación de líneas y reducción en los niveles de satisfacción del usuario [5].

En los últimos años, el país ha impulsado iniciativas orientadas a la transformación digital, destacando la digitalización de trámites, el uso de servicios telemáticos y la adopción progresiva de tecnologías basadas en inteligencia artificial. Sin embargo, a pesar de estos avances, muchas organizaciones ecuatorianas aún no han implementado soluciones de automatización conversacional que permitan optimizar su atención al cliente. En este sentido, Issabel, ampliamente utilizado en Ecuador debido a su carácter open source y su compatibilidad con infraestructuras de telefonía IP, representa una plataforma con gran potencial para incorporar herramientas inteligentes que mejoren la eficiencia del servicio [6].

La introducción de un asistente conversacional basado en inteligencia artificial permitiría automatizar la resolución de consultas básicas, liberando a los agentes para atender casos que

requieren criterio humano y reduciendo los costos operativos asociados al manejo manual de llamadas. Asimismo, contribuiría a mejorar la experiencia de los ciudadanos y clientes al ofrecer respuestas rápidas, uniformes y disponibles en cualquier momento. Esta innovación es especialmente relevante en un contexto nacional donde la demanda de servicios de información continúa creciendo y donde se requiere optimizar los recursos tecnológicos y humanos disponibles [7].

La investigación adquiere relevancia porque articula áreas tecnológicas emergentes como el procesamiento de lenguaje natural, los sistemas conversacionales para generar conocimiento práctico y directamente aplicable a las necesidades actuales del país. Asimismo, el desarrollo de un prototipo funcional integrado a Issabel Call Center puede constituirse en un referente técnico para instituciones públicas, empresas privadas y emprendedores ecuatorianos interesados en modernizar y optimizar sus canales de atención al usuario [8].

Capítulo II: Marco teórico y fundamentos de la investigación

El presente capítulo desarrolla los fundamentos teóricos que sustentan la propuesta de implementación de un asistente conversacional basado en inteligencia artificial integrado a un sistema de call center. En este sentido, se abordan conceptos relacionados con la inteligencia artificial, el procesamiento de lenguaje natural, los modelos de lenguaje, los sistemas conversacionales y la arquitectura de plataformas de telefonía como Issabel. La comprensión de estos elementos resulta fundamental para el diseño, desarrollo e implementación de soluciones automatizadas orientadas a la atención de consultas básicas en entornos empresariales.

2.1. Inteligencia Artificial y sus Fundamentos

La inteligencia artificial (IA) es una rama de la informática que se ocupa del desarrollo de sistemas capaces de realizar tareas que normalmente requieren inteligencia humana, tales como el razonamiento, el aprendizaje, la percepción y la comprensión del lenguaje natural (AWS, 2024). En las últimas décadas, la IA ha evolucionado de sistemas basados exclusivamente en reglas hacia modelos estadísticos y de aprendizaje profundo, lo cual ha permitido una expansión significativa en sus aplicaciones prácticas dentro de la industria, la educación, la salud y la atención al cliente.

Según Garzón-Quiroz et al. (2025), los asistentes conversacionales impulsados por IA han demostrado ser altamente eficientes en la gestión de procesos empresariales, posibilitando la automatización de servicios al cliente, la reducción de costos operativos y la mejora de la satisfacción del usuario mediante disponibilidad continua y personalización de las respuestas. Esta capacidad de adaptación ha convertido a la IA en una herramienta estratégica para

organizaciones que buscan optimizar sus canales de comunicación, especialmente en contextos donde el volumen de interacciones es elevado y la repetición de consultas es frecuente.

En el contexto latinoamericano y ecuatoriano, la adopción de tecnologías de IA en los servicios de atención al cliente ha comenzado a ganar terreno de manera progresiva, impulsada por iniciativas de transformación digital promovidas tanto desde el sector privado como desde las políticas públicas. El Ministerio de Telecomunicaciones y de la Sociedad de la Información del Ecuador (2025) ha señalado que el gobierno nacional impulsa programas como EmpodéraTech Ecuador, orientados a fomentar la adopción de tecnologías emergentes en distintos sectores productivos e institucionales del país.

2.2. Procesamiento de Lenguaje Natural (PLN)

El Procesamiento de Lenguaje Natural (PLN), conocido por sus siglas en inglés como NLP (Natural Language Processing), es una subdisciplina de la inteligencia artificial que se ocupa de la interacción entre las computadoras y el lenguaje humano. Su objetivo principal es reducir la brecha de comunicación entre las máquinas y los seres humanos, permitiendo que los sistemas computacionales comprendan, interpreten y generen texto o voz de manera coherente y contextualmente pertinente [9].

La evolución del PLN ha estado marcada por hitos tecnológicos significativos. Durante la década de 1990 y principios de los 2000, los avances se concentraron en aplicaciones como el filtrado de correo no deseado y la clasificación de documentos. Sin embargo, la introducción de la arquitectura Transformer en 2017 representó un punto de inflexión, al permitir el procesamiento en paralelo de secuencias de texto y el manejo de memoria a largo plazo mediante mecanismos de atención [9]. Esta arquitectura sentó las bases para el desarrollo de modelos de

lenguaje de gran escala como BERT, GPT y Gemini, que hoy dominan el panorama de la IA conversacional.

De acuerdo con la revisión sistemática realizada por Giraldo Forero y Orozco Duque (2023), la evolución del PLN puede comprenderse a través de tres grandes paradigmas: el enfoque basado en reglas lingüísticas, el enfoque estadístico y el enfoque basado en redes neuronales profundas. Este último, impulsado por arquitecturas transformer y técnicas de preentrenamiento a gran escala, ha logrado superar ampliamente a sus predecesores en tareas como comprensión de textos, traducción automática, clasificación de intenciones y generación de respuestas contextualizadas.

Las tareas fundamentales que abarca el PLN incluyen el análisis morfológico y sintáctico, la comprensión semántica, la pragmática del lenguaje, el reconocimiento de entidades nombradas, la clasificación de intenciones y la generación de lenguaje natural (NLG). En el contexto de los asistentes conversacionales para centros de atención telefónica, las tareas más relevantes son la clasificación de intenciones —que permite identificar qué desea el usuario— y la extracción de entidades —que permite detectar datos específicos mencionados en la consulta—, como fechas, nombres o números de referencia [10].

2.2.1. Modelos de Clasificación de Intenciones

La clasificación de intenciones es un proceso mediante el cual un sistema de PLN analiza el texto o la transcripción de voz de un usuario y lo asigna a una categoría predefinida que representa su propósito comunicativo. Este proceso es fundamental en los asistentes conversacionales, ya que permite determinar qué acción debe ejecutarse o qué respuesta debe generarse en función de lo que el usuario desea obtener[11].

Los enfoques modernos para la clasificación de intenciones se basan principalmente en modelos de aprendizaje automático supervisado entrenados con ejemplos de consultas etiquetadas. Técnicas como la codificación de oraciones mediante embeddings semánticos, combinadas con clasificadores como redes neuronales convolucionales o transformadores ajustados finamente (fine-tuned), han demostrado altos niveles de precisión en entornos de producción real [10]. La selección del modelo adecuado depende de factores como el tamaño del corpus de entrenamiento, la cantidad de intenciones a clasificar, el idioma de las consultas y los recursos computacionales disponibles.

2.3. Grandes Modelos de Lenguaje (LLM)

Los Grandes Modelos de Lenguaje, conocidos como LLM por sus siglas en inglés (Large Language Models), son sistemas de inteligencia artificial entrenados sobre enormes volúmenes de datos textuales con el propósito de comprender y generar lenguaje humano de manera coherente y contextualmente pertinente. Según Amazon Web Services (2024), los LLM son capaces de ejecutar tareas completamente distintas a partir de un único modelo, incluyendo responder preguntas, resumir documentos, traducir idiomas y completar secuencias de texto. Su flexibilidad y potencia los convierten en la base tecnológica de la generación actual de asistentes virtuales e interfaces conversacionales.

Entre los LLM más destacados se encuentran GPT (OpenAI), Gemini (Google DeepMind) y Claude (Anthropic). Según IBM (2024), Gemini es una familia de modelos multimodales desarrollada por Google DeepMind, cuya primera versión fue anunciada el 6 de diciembre de 2023. Desde su lanzamiento, Google puso a disposición de los desarrolladores acceso a estos modelos a través de la API de Gemini en Google AI Studio y Google Cloud Vertex AI, facilitando su integración en aplicaciones de terceros. La familia Gemini contempla

variantes como Gemini Pro, Flash y Ultra, optimizadas para distintos casos de uso, desde aplicaciones de baja latencia hasta tareas de razonamiento complejo.

Para el proyecto que sustenta la presente investigación, la API de Gemini resulta especialmente relevante al permitir la integración de capacidades avanzadas de PLN dentro de la arquitectura del asistente conversacional integrado al Issabel Call Center. A través de esta API, es posible enviar transcripciones de voz —previamente generadas por el módulo de reconocimiento de voz— y obtener respuestas generadas de manera dinámica, coherente y contextualizada, superando las limitaciones de los sistemas IVR tradicionales basados en menús predefinidos[12].

2.3.1. IA Generativa y Asistentes Conversacionales

La inteligencia artificial generativa representa una rama de los modelos de lenguaje orientada a la creación autónoma de contenido —ya sea texto, imágenes, código o audio— a partir de instrucciones en lenguaje natural. En el ámbito de los asistentes conversacionales, la IA generativa permite que los sistemas produzcan respuestas dinámicas, adaptadas al contexto de cada interacción, en lugar de limitarse a respuestas predefinidas y estáticas [13].

Según Garzón-Quiroz et al. (2025), los chatbots impulsados por LLM están diseñados para interactuar con los usuarios mediante el uso de PLN, pudiendo comprender y generar texto de manera coherente y respondiendo a consultas en tiempo real. Estos sistemas utilizan técnicas avanzadas de NLP y están entrenados en grandes cantidades de datos para mejorar su precisión y relevancia en las respuestas. En el contexto del servicio al cliente, la IA generativa permite manejar consultas no previstas explícitamente durante el diseño del sistema, dotando al asistente de una flexibilidad significativamente mayor que los enfoques basados en reglas o en árboles de decisión.

2.4. Reconocimiento Automático de Voz (ASR) y Síntesis de Voz (TTS)

En sistemas de atención telefónica automatizada, la interacción con el usuario se produce mediante la voz, lo cual hace indispensable contar con tecnologías de reconocimiento automático del habla (ASR, por sus siglas en inglés: Automatic Speech Recognition) y de síntesis de texto a voz (TTS: Text-to-Speech). El ASR convierte la señal de audio captada durante una llamada en texto procesable por los módulos de PLN, mientras que el TTS transforma las respuestas generadas por el modelo de IA en señales de audio que son reproducidas al usuario.

Dentro de las herramientas de ASR de código abierto disponibles actualmente, Vosk se destaca como una de las más versátiles y eficientes para entornos con recursos limitados. De acuerdo con Alphacephei (2023), Vosk es un toolkit de reconocimiento de voz fuera de línea que soporta más de 20 idiomas y dialectos, incluyendo español, con modelos ligeros de aproximadamente 50 MB que permiten una transcripción continua de vocabulario amplio, respuesta de latencia cero mediante API de streaming y reconfiguración del vocabulario sin necesidad de reentrenamiento. Esta característica lo convierte en una solución viable para la integración dentro de servidores locales en entornos de call center, donde la privacidad de los datos y la baja latencia son requisitos fundamentales.

De manera complementaria, el módulo de síntesis de voz del sistema propuesto opera mediante un servidor Flask que implementa tecnología TTS para convertir las respuestas generadas por el LLM en señales de audio reproducibles. Esta arquitectura distribuida, donde cada componente del sistema opera como un microservicio independiente, permite una mayor flexibilidad, escalabilidad y facilidad de mantenimiento frente a soluciones monolíticas [15].

2.5. Arquitectura del Issabel Call Center

Issabel es una plataforma de telefonía IP y comunicaciones unificadas de código abierto, basada en Asterisk, que permite gestionar de manera integral la comunicación empresarial. Según Netelip (2021), Issabel es un software de código abierto de telefonía IP y comunicaciones unificadas basado en Asterisk, que permite unificar de forma integral todas las comunicaciones de una empresa; es accesible como servicio alojado en la nube, disponible las 24 horas del día durante los 365 días del año.

El módulo de Call Center de Issabel proporciona funcionalidades avanzadas para la gestión de llamadas entrantes y salientes, incluyendo la administración de colas de espera, el enrutamiento inteligente de llamadas, el monitoreo en tiempo real de los agentes, la grabación de llamadas y la generación de reportes estadísticos detallados [15]. Su carácter open source y su compatibilidad con infraestructuras de telefonía IP ampliamente utilizadas en Ecuador lo posicionan como una plataforma con gran potencial para incorporar herramientas inteligentes que mejoren la eficiencia del servicio [16].

En versiones recientes, Issabel ha incorporado soporte nativo para la integración con motores LLM a través de su módulo Chatbroker, el cual permite conectar el sistema con proveedores de inteligencia artificial como OpenAI, Anthropic, DeepSeek y Groq, entre otros. Esto posibilita la implementación de chatbots inteligentes y asistentes virtuales capaces de automatizar respuestas, gestionar consultas frecuentes y desarrollar flujos conversacionales dinámicos tanto en canales de voz como en canales digitales[15].

2.6. Telefonía IP y Protocolo VoIP

La telefonía IP (Voice over Internet Protocol, VoIP) es una tecnología que permite transmitir señales de voz a través de redes de datos basadas en el protocolo IP, en lugar de la red

telefónica conmutada tradicional (PSTN). Esta tecnología ha revolucionado las comunicaciones empresariales al ofrecer mayor flexibilidad, escalabilidad y reducción de costos frente a las soluciones telefónicas convencionales. En Ecuador, la adopción de soluciones VoIP como Issabel se ha extendido tanto en el sector privado como en instituciones públicas, impulsada por su carácter open source y su compatibilidad con infraestructuras de red existentes.

El Protocolo de Inicio de Sesión (SIP) es el estándar más ampliamente utilizado para la señalización en sistemas VoIP. SIP gestiona el establecimiento, modificación y finalización de sesiones de comunicación, incluyendo llamadas de voz, videollamadas y mensajería. En el contexto de Issabel, SIP opera como el protocolo mediante el cual las extensiones —ya sean teléfonos físicos, softphones o aplicaciones— se registran en el servidor y establecen comunicación entre sí o hacia el exterior a través de troncos SIP[15].

La integración de capacidades de inteligencia artificial dentro de entornos VoIP representa un avance significativo respecto a los sistemas IVR (Interactive Voice Response) tradicionales, que se limitaban a menús de opciones numéricas y respuestas de texto a voz sin capacidad de comprensión del lenguaje natural. Como señalan Carrión Vega et al. (2025), los sistemas modernos de atención al cliente requieren soluciones que no solo gestionen el enrutamiento de llamadas, sino que también sean capaces de interpretar las solicitudes del usuario y generar respuestas pertinentes de manera autónoma[16].

2.6.1. Asterisk y la Interfaz AGI

Asterisk es el motor de telefonía IP sobre el cual se construye Issabel. Se trata de uno de los frameworks de VoIP más potentes y flexibles del mundo, cuya arquitectura permite a los desarrolladores acceder a sus funcionalidades internas mediante tres interfaces nativas: AMI

(Asterisk Manager Interface), AGI (Asterisk Gateway Interface) y ARI (Asterisk REST Interface) [15].

La interfaz AGI (Asterisk Gateway Interface) es el mecanismo mediante el cual se conecta el asistente conversacional con el motor de telefonía. AGI permite ejecutar scripts externos —escritos en lenguajes como Python o PHP— directamente desde el dialplan de Asterisk, habilitando así la comunicación bidireccional entre la lógica del call center y los módulos de procesamiento de inteligencia artificial. A través de AGI, es posible capturar la entrada de voz del usuario, invocar al módulo de reconocimiento de voz, enviar la transcripción al modelo LLM y reproducir la respuesta generada mediante síntesis de voz, todo dentro del flujo de una llamada telefónica activa.

El protocolo SIP (Session Initiation Protocol) es el estándar mediante el cual las extensiones se registran y comunican dentro del sistema Issabel. El puerto 2001 es utilizado por Issabel Asterisk para la gestión de comunicaciones SIP, permitiendo que los dispositivos de usuario final actúen como medios de interacción que capturan la entrada de voz y reproducen las respuestas generadas, mientras que el procesamiento de intenciones, la generación de respuestas y la integración con el sistema Issabel se realizan en servicios backend [17]

2.7. Asistentes Conversacionales en Centros de Atención

Los asistentes conversacionales —también denominados chatbots o voicebots según el canal de interacción— son sistemas de software que simulan una conversación con el usuario mediante el uso de lenguaje natural. Su evolución desde los primeros sistemas basados en reglas predefinidas hacia los actuales modelos impulsados por LLM ha transformado radicalmente la forma en que las organizaciones diseñan e implementan sus canales de atención al cliente[18].

Según Garzón-Quiroz et al. (2025), los chatbots basados en reglas predefinidas son programas que siguen instrucciones fijas para interactuar con los usuarios, y resultan especialmente útiles en el servicio al cliente para manejar consultas repetitivas y estructuradas de manera eficiente. Sin embargo, su incapacidad para comprender el contexto y las variaciones del lenguaje puede resultar frustrante para los usuarios cuando las consultas no coinciden exactamente con las reglas predefinidas, limitando la efectividad del sistema en entornos dinámicos. En contraste, los asistentes basados en LLM comprenden y generan texto de manera coherente, respondiendo a consultas en tiempo real con precisión y relevancia contextual.

En los call centers, la introducción de asistentes conversacionales basados en IA ha demostrado generar beneficios concretos y cuantificables. Ben Ammar y Ben Ammar (2021) señalan que la automatización basada en IA en el servicio al cliente mejora el tiempo de respuesta y la eficiencia operativa, reduciendo la carga de trabajo repetitivo de los agentes humanos. Por su parte, el mercado global de IA para centros de llamadas fue valorado en 1.740 millones de dólares en 2024, con una tasa de crecimiento anual compuesta proyectada del 23.67% hasta 2032, lo que refleja la rápida adopción de estas soluciones a nivel mundial[19].

La automatización de consultas básicas mediante asistentes conversacionales libera a los agentes humanos para atender casos que requieren criterio, empatía o conocimiento especializado, lo que redundará en una mejor distribución de los recursos humanos disponibles y en una reducción de los costos operativos asociados al manejo manual de llamadas[20]. Además, la disponibilidad ininterrumpida del asistente conversacional —operativo las 24 horas del día durante los 7 días de la semana— elimina las barreras temporales del servicio de atención, mejorando significativamente la experiencia del usuario final.

2.8. Transformación Digital en el Sector de Atención al Cliente en Ecuador

Ecuador ha experimentado un proceso progresivo de transformación digital en los últimos años, especialmente en el ámbito de los servicios públicos y la atención al ciudadano. El Ministerio de Telecomunicaciones y de la Sociedad de la Información (2025) ha impulsado iniciativas orientadas a la digitalización de trámites, el uso de servicios telemáticos y la adopción de tecnologías basadas en inteligencia artificial. Sin embargo, a pesar de estos avances, muchas organizaciones ecuatorianas aún no han implementado soluciones de automatización conversacional que permitan optimizar su atención al cliente[21].

Los centros de atención telefónica como la que se visualiza en la Figura 2 continúan siendo un componente fundamental para la comunicación entre instituciones públicas, empresas privadas y usuarios. Valarezo Pardo y Cárdenas Villavicencio (2025) señalan que gran parte de estos sistemas continúa operando bajo modelos tradicionales, donde la atención inicial depende exclusivamente de agentes humanos, generando limitaciones operativas —especialmente en entornos de alto volumen de llamadas— como tiempos de espera prolongados, saturación de líneas y reducción en los niveles de satisfacción del usuario.

Figura 2

Call center de la empresa Bangara-Quito



La plataforma Issabel, ampliamente utilizada en Ecuador por su carácter open source y su compatibilidad con infraestructuras de telefonía IP, representa una base tecnológica sólida para la incorporación de herramientas de inteligencia artificial. El desarrollo de un prototipo funcional integrado a Issabel Call Center puede constituirse en un referente técnico para instituciones públicas, empresas privadas y emprendedores ecuatorianos interesados en modernizar y optimizar sus canales de atención al usuario [23]

2.9. Arquitectura del Sistema Propuesto

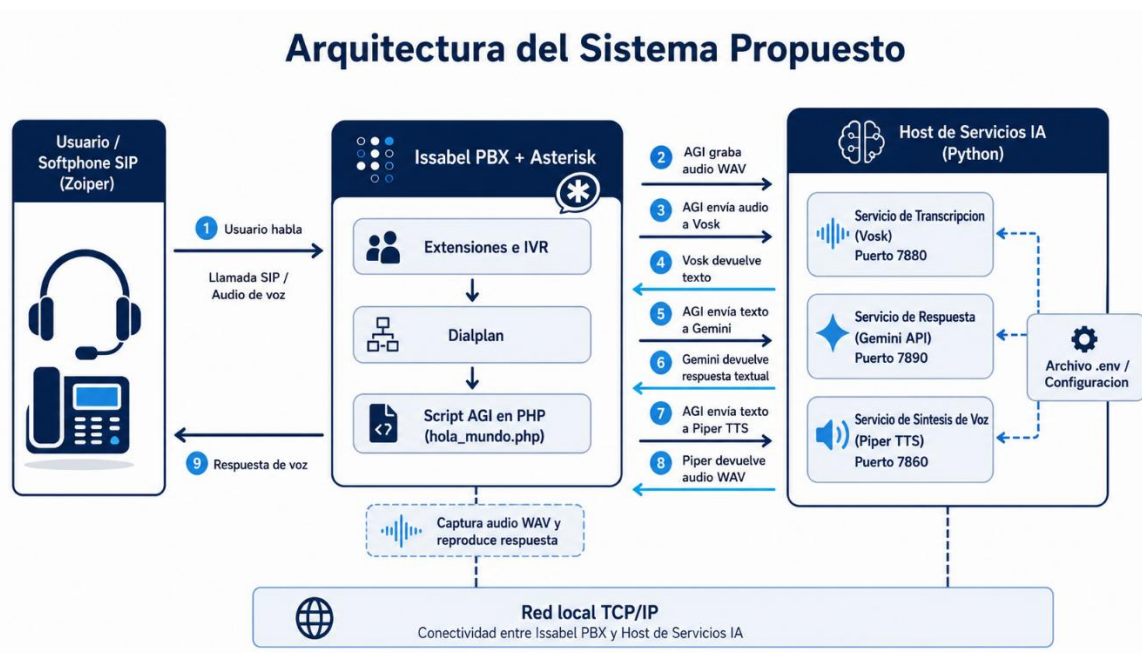
La arquitectura del asistente conversacional propuesto integra múltiples componentes tecnológicos que operan de manera coordinada para gestionar el ciclo completo de una interacción por voz dentro del Issabel Call Center. El diagrama que se muestra en la Figura 3 de desarrollo del sistema contempla los siguientes elementos principales: el servidor Issabel con Asterisk, el servidor de transcripción de voz a texto (Vosk), el servidor con la API de Gemini

para la resolución de consultas mediante LLM, y el servidor TTS (Flask) para la generación de respuestas de voz[26].

El flujo de interacción se inicia cuando el dispositivo del usuario final —un teléfono SIP o softphone— establece una llamada hacia el Issabel Call Center. El servidor Asterisk recibe la llamada y, mediante el dialplan, invoca un script AGI que actúa como orquestador del proceso. Este script captura el audio de la llamada y lo envía al servidor Vosk para su transcripción. El texto resultante es enviado a la API de Gemini, que genera una respuesta pertinente en lenguaje natural. Dicha respuesta es entonces enviada al servidor TTS Flask para su conversión a audio, que finalmente es reproducido al usuario a través de la llamada activa[25].

Figura 3

Arquitectura del sistema propuesto



Esta arquitectura distribuida en microservicios garantiza la independencia funcional de cada componente, facilitando su mantenimiento, actualización y escalabilidad. El dispositivo del usuario final actúa únicamente como un medio de interacción, capturando la entrada de voz y

reproduciendo las respuestas generadas, mientras que el procesamiento de intenciones, la generación de respuestas y la integración con el sistema Issabel se realizan en servicios backend [27].

Capítulo III: Levantamiento de requerimientos y diseño

En este capítulo se presenta la fundamentación técnica del sistema propuesto, concentrándose en los requerimientos definidos, la selección de hardware y software, la elección de lenguajes de programación y la justificación de cada componente utilizado. A diferencia del capítulo siguiente, orientado a la implementación y validación práctica, el presente capítulo organiza las decisiones de diseño que hicieron posible la construcción de la arquitectura conversacional basada en telefonía IP e inteligencia artificial.

La solución planteada tuvo como propósito transformar una interacción telefónica tradicional en un flujo conversacional capaz de recibir una consulta oral, interpretarla y devolver una respuesta sintetizada dentro de la misma llamada. Para lograrlo fue necesario establecer, antes de la implementación, un conjunto de criterios de selección tecnológica que garantizara viabilidad técnica, compatibilidad entre componentes, facilidad de pruebas y posibilidad de ampliación futura.

3.1 Metodología de Desarrollo

El diseño de una solución de respuesta de voz interactiva no depende únicamente de la programación de sus módulos, sino también de la definición previa de las capacidades que el sistema debe satisfacer. En este sentido, el capítulo se estructura como una fase de análisis y decisión tecnológica, donde cada recurso fue evaluado a partir de su función dentro de la

arquitectura general, su facilidad de integración y su pertinencia para un contexto académico de experimentación.

La elección de las herramientas no respondió a una preferencia arbitraria. Cada componente fue seleccionado para resolver una necesidad concreta: gestionar la llamada, grabar audio, transcribir voz, generar una respuesta textual y sintetizar dicha respuesta nuevamente en formato de voz. La relación entre requerimiento y componente constituye, por tanto, el eje central del presente capítulo.

3.2 Requerimientos del sistema

Los requerimientos que se muestran en la Tabla 1 fueron establecidos a partir del objetivo general del proyecto: construir un prototipo IVR conversacional capaz de operar sobre una infraestructura PBX. Para ello se definieron requerimientos funcionales, enfocados en las capacidades visibles del sistema, y requerimientos no funcionales.

Tabla 1

Requerimientos funcionales y no funcionales del sistema propuesto

Código	Tipo	Requerimiento	Justificación
RF-01	Funcional	Registrar extensiones SIP y permitir llamadas internas entre usuarios.	Era indispensable validar primero la infraestructura de telefonía antes de integrar componentes de inteligencia artificial.
RF-02	Funcional	Recibir una llamada en un número IVR y transferir el control al script AGI.	Permitió disponer de un punto de entrada único para el flujo conversacional.

RF-03	Funcional	Grabar la consulta oral del usuario en formato WAV.	La grabación fue necesaria para alimentar el servicio de transcripción con un formato estable y reutilizable.
RF-04	Funcional	Transcribir la voz del usuario a texto en español.	La consulta debía convertirse en texto para ser procesada por un modelo generativo.
RF-05	Funcional	Generar una respuesta breve y coherente a partir del texto transcrito.	La solución buscó responder preguntas abiertas, no solo navegar menús DTMF.
RF-06	Funcional	Transformar la respuesta textual en audio reproducible por Asterisk.	Sin síntesis de voz no era posible cerrar el ciclo conversacional dentro de la llamada.
RF-07	Funcional	Mantener la conversación activa hasta que el usuario finalice con una frase de salida.	Se buscó superar el modelo de una sola interacción por llamada.
RNF-01	No funcional	Trabajar con componentes de despliegue local o controlado.	Esto redujo dependencia externa, mejoró el control del entorno y facilitó pruebas repetibles.

RNF-02	No funcional	Usar formatos compatibles con telefonía, especialmente audio a 8 kHz.	La compatibilidad con Asterisk exigió estandarizar el audio sintetizado.
RNF-03	No funcional	Separar el sistema en módulos independientes conectados por HTTP.	La modularidad simplificó mantenimiento, pruebas unitarias e integración progresiva.

La definición de estos requerimientos permitió ordenar el proceso de selección tecnológica. En lugar de escoger herramientas de forma aislada, se partió de lo que el sistema debía hacer y, a partir de ello, se identificaron los recursos más adecuados para cumplir cada función. Este enfoque evitó sobredimensionar la solución y favoreció la construcción de una arquitectura modular.

3.3 Requerimientos de hardware

En términos de hardware, la solución no exigió infraestructura especializada de alto costo, pero sí una distribución clara de responsabilidades entre los equipos disponibles. Se definió un nodo principal para la central telefónica y un nodo auxiliar para los servicios de procesamiento, lo que permitió separar la lógica PBX de las tareas de reconocimiento de voz, generación de respuestas y síntesis como se indica en la Tabla 2.

Tabla 2

Requerimientos de hardware considerados en la solución

Recurso	Descripción	Uso en el proyecto	Justificación de selección
---------	-------------	--------------------	----------------------------

Servidor virtual PBX	Máquina virtual destinada a Issabel y Asterisk.	Administración de extensiones, IVR y ejecución del AGI.	Se eligió una máquina virtual para aislar la central telefónica y facilitar reinicios, snapshots y pruebas controladas.
Equipo anfitrión	Computador con Windows y capacidad para ejecutar Python, Vosk, Gemini y Piper.	Hospedaje de servicios STT, IA y TTS.	Se aprovechó el equipo disponible con suficiente capacidad de memoria y procesamiento.
Red local	Conectividad IP entre 192.168.0.104 y 192.168.0.102.	Intercambio HTTP entre PBX y microservicios.	La comunicación en red local redujo latencias y permitió depuración directa entre nodos.
Softphones	Terminales SIP sobre Zoiper y configuración equivalente en móviles.	Pruebas de registro, llamadas internas y acceso al IVR.	Se eligieron por su facilidad de configuración y por permitir pruebas rápidas sin teléfonos IP físicos.

La decisión de utilizar una máquina virtual para Issabel respondió a criterios de flexibilidad y control experimental. Esto facilitó la configuración de la PBX, la restauración de estados previos y la administración del entorno sin comprometer el equipo anfitrión. De la misma manera, el uso de un computador independiente para los servicios Python permitió descargar de la PBX el procesamiento asociado al reconocimiento de voz.

3.4 Requerimientos de software

Fue necesario disponer de una plataforma PBX, un motor de telefonía, un lenguaje para la lógica de integración, bibliotecas para exposición de servicios web y motores específicos para voz, texto e inteligencia artificial. La selección se realizó priorizando compatibilidad, estabilidad, facilidad de despliegue y disponibilidad de documentación oficial como se detalla en la Tabla 3.

Tabla 3

Software seleccionado para la arquitectura conversacional

Componente	Software	Uso	Motivo de elección	Resultado esperado
PBX	Issabel	Gestión web de la central telefónica.	Integra una administración práctica sobre Asterisk y facilita la configuración de extensiones e IVR.	Reducir complejidad operativa en la configuración inicial.

Motor de telefonía	Asterisk 16.7.0	Señalización SIP, grabación y reproducción de audio.	Es un estándar abierto ampliamente documentado y compatible con AGI.	Disponer de una base sólida para el control de llamadas.
Script de integración	PHP con phpagi	Intermediar entre la llamada y los servicios externos.	AGI en PHP resultó directo de integrar en Issabel y coherente con el material de referencia del proyecto.	Gestionar lógica de llamada, grabación y reproducción.
Transcripción	Vosk	Reconocimiento de voz en español.	Permite operación local, modelos descargables y un consumo razonable.	Obtener texto a partir del audio del usuario.
Modelo generativo	Gemini API	Producción de respuestas	Ofrece una API sencilla para consultas	Responder preguntas abiertas de

		breves en lenguaje natural.	textuales y respuesta rápida orientada a integración.	forma conversacional.
Síntesis de voz	Piper	Conversión de texto en audio WAV.	Permite síntesis local, voces configurables y buena integración con archivos WAV.	Cerrar el ciclo voz-texto-voz dentro de la llamada.
Servicios HTTP	Flask y Waitress	Exposición de endpoints REST para STT, IA y TTS.	Flask simplifica el desarrollo y Waitress ofrece una ejecución estable para servicios ligeros.	Separar cargas y facilitar pruebas individuales.
Cliente SIP	Zoiper	Pruebas de registro y llamadas.	Es simple de configurar, ampliamente utilizado y útil en entornos académicos.	Verificar que la telefonía interna funcionara antes de integrar IA.

El conjunto resultante respondió a una lógica de capas. Issabel y Asterisk conformaron la capa de telefonía; PHP y phpagi actuaron como capa de llamadas; y Python, junto con sus bibliotecas, soportó la capa de servicios cognitivos. Este desacoplamiento redujo el riesgo de dependencias rígidas y facilitó la validación independiente.

3.5 Elección del servidor PBX y del motor de telefonía

La elección de Issabel como plataforma PBX obedeció a la necesidad de contar con una interfaz de administración accesible para la creación de extensiones, rutas y módulos IVR, sin renunciar a la potencia de Asterisk como motor de telefonía. Issabel aportó un entorno de configuración más amigable para el desarrollo del proyecto, mientras que Asterisk ofreció el control real de la llamada, la grabación y la ejecución de aplicaciones como AGI y Playback (Issabel, n.d.; The Asterisk Project, n.d.).

Esta combinación fue adecuada porque permitió dividir las tareas entre administración y ejecución. Desde Issabel se realizaron las configuraciones PBX necesarias, y desde Asterisk se explotaron los mecanismos de bajo nivel indispensables para la automatización de la llamada. La selección del servidor, por tanto, no se justificó solo por disponibilidad, sino por su idoneidad para una arquitectura experimental con integración externa.

3.6 Elección de los lenguajes de programación

La solución utilizó más de un lenguaje de programación que se detalla en la Tabla 4, debido a la naturaleza del problema. PHP se reservó para el script AGI porque su integración con la biblioteca phpagi permitió interactuar con Asterisk de manera directa, reproduciendo audios, contestando llamadas, grabando mensajes y gestionando eventos del flujo telefónico. Además, esta elección guardó coherencia con los archivos base disponibles durante el desarrollo del proyecto.

Python se seleccionó para los servicios de transcripción, inteligencia generativa y síntesis de voz por la madurez de su ecosistema en aplicaciones de IA y procesamiento de audio. La disponibilidad de bibliotecas como vosk, google-genai, scipy y soundfile redujo la complejidad de implementación y permitió concentrar el desarrollo en la lógica del proyecto, en lugar de construir herramientas auxiliares desde cero.

Tabla 4

Lenguajes y marcos de desarrollo seleccionados

Lenguaje / marco	Dónde se utilizó	Razón de elección	Qué permitió realizar
PHP	Script AGI ejecutado por Asterisk.	Se integró de forma natural con phpagi y con el flujo tradicional de scripts en Issabel.	Controlar la llamada, ejecutar grabaciones, invocar servicios web y reproducir respuestas.
Python	Servicios de transcripción, Gemini y TTS.	Su ecosistema dispone de bibliotecas maduras para IA, audio y servicios web.	Desarrollar rápidamente microservicios separados y fáciles de depurar.
Flask	Capa HTTP de los servicios Python.	Ofreció una curva de aprendizaje baja y una estructura mínima para endpoints REST.	Exponer rutas como /transcribe, /gemini y /convert.
Waitress	Publicación de los servicios en puertos locales.	Aportó un servidor WSGI sencillo y	Mantener los servicios disponibles para las llamadas de Issabel.

estable para pruebas

continuas.

La convivencia de PHP y Python no representó una desventaja, sino una distribución funcional del problema. PHP quedó asociado a la central y a la llamada, mientras que Python se destinó a los servicios especializados. Esta separación fue consistente con una arquitectura orientada a servicios, donde cada lenguaje se empleó en el contexto para el que ofrecía mejores ventajas.

3.7 Elección del modelo de transcripción de voz

Vosk fue seleccionado como motor de reconocimiento automático del habla debido a su capacidad de funcionamiento local y a la disponibilidad de modelos para idioma español. En el proyecto, esta característica fue especialmente valiosa porque permitió realizar pruebas repetibles sin depender de un servicio de transcripción en la nube. A ello se sumó su integración sencilla con Python y la posibilidad de trabajar con archivos WAV generados por la propia PBX (Alpha Cephei, n.d.).

La justificación principal de esta elección fue técnica y metodológica. Vosk proporcionó una solución suficientemente ligera para un entorno de laboratorio, al operar localmente, facilitó la depuración del sistema y eliminó variables externas como cambios de API, limitaciones de conectividad o costos por consumo. Su función dentro del proyecto consistió en transformar cada mensaje grabado por Asterisk en texto utilizable por el motor generativo.

3.8 Elección del modelo generativo para la respuesta

Para la generación de respuestas se optó por la API de Gemini. La razón principal fue la disponibilidad de un servicio accesible mediante solicitudes HTTP, con capacidad de producir

respuestas breves y comprensibles en español. Esta elección permitió implementar una capa de interpretación semántica que superó el comportamiento tradicional de un IVR basado exclusivamente en rutas fijas o menús DTMF (Google, n.d.).

La justificación de Gemini se relacionó con la necesidad de introducir flexibilidad conversacional. Mientras que la PBX resuelve el transporte de la llamada y Vosk obtiene el contenido textual, el modelo generativo aporta la construcción de la respuesta. En consecuencia, Gemini fue utilizado como núcleo de razonamiento textual del sistema. Su función específica fue recibir una consulta transcrita y devolver una respuesta breve, clara y adecuada para ser leída posteriormente por un sintetizador de voz.

3.9 Elección del sintetizador de voz

Piper fue elegido para la síntesis de voz porque permite generar audio de manera local y trabajar con modelos de voz en formato ONNX. Esta característica resultó conveniente en el proyecto, ya que redujo la dependencia de servicios externos y ofreció control sobre las voces disponibles, el formato de salida y la integración posterior con Asterisk (OHF-Voice, n.d.).

La función de Piper dentro de la arquitectura fue convertir la respuesta textual generada por Gemini en un archivo WAV apto para reproducción telefónica. La justificación de su selección se apoyó en tres factores: operación local, adaptabilidad a diferentes voces y facilidad para reprocesar el audio a 8 kHz, condición necesaria para una integración robusta con la central telefónica.

3.10 Elección del modelo de despliegue y del servidor de servicios

Los servicios auxiliares se diseñaron como microservicios HTTP independientes, expuestos mediante Flask y publicados con Waitress. Esta decisión respondió al objetivo de desacoplar el sistema en unidades pequeñas y verificables. Cada servicio escuchó en un puerto

específico: 7880 para la transcripción, 7890 para el motor generativo y 7860 para la síntesis. De este modo, el AGI pudo actuar como un conjunto que consume servicios especializados en lugar de concentrar toda la lógica en un solo proceso.

La justificación del modelo cliente-servidor fue doble. Por un lado, permitió separar la capa telefónica de la capa de procesamiento inteligente, evitando sobrecargar la PBX. Por otro lado, facilitó las pruebas individuales con endpoints de salud y conversión, haciendo posible detectar fallas por servicio. La elección de Flask y Waitress se sustentó en su bajo nivel de complejidad, su rápida configuración y su idoneidad para entornos académicos y de prototipado.

3.11 Elección del cliente SIP para pruebas

La utilización de Zoiper como softphone respondió a la necesidad de disponer de terminales SIP rápidos de configurar y funcionales para pruebas de laboratorio. Su papel no fue estructural dentro de la arquitectura final, pero sí metodológicamente importante, porque permitió verificar el registro de extensiones, la correcta marcación interna y el acceso al IVR antes de integrar las capacidades de inteligencia artificial.

La justificación de esta elección radicó en que un sistema telefónico conversacional debe probarse desde la experiencia real de usuario. Por ello, contar con clientes SIP funcionales fue un requisito previo para comprobar la operatividad de las extensiones, la calidad del audio y el comportamiento del flujo de llamada en condiciones cercanas a uso real.

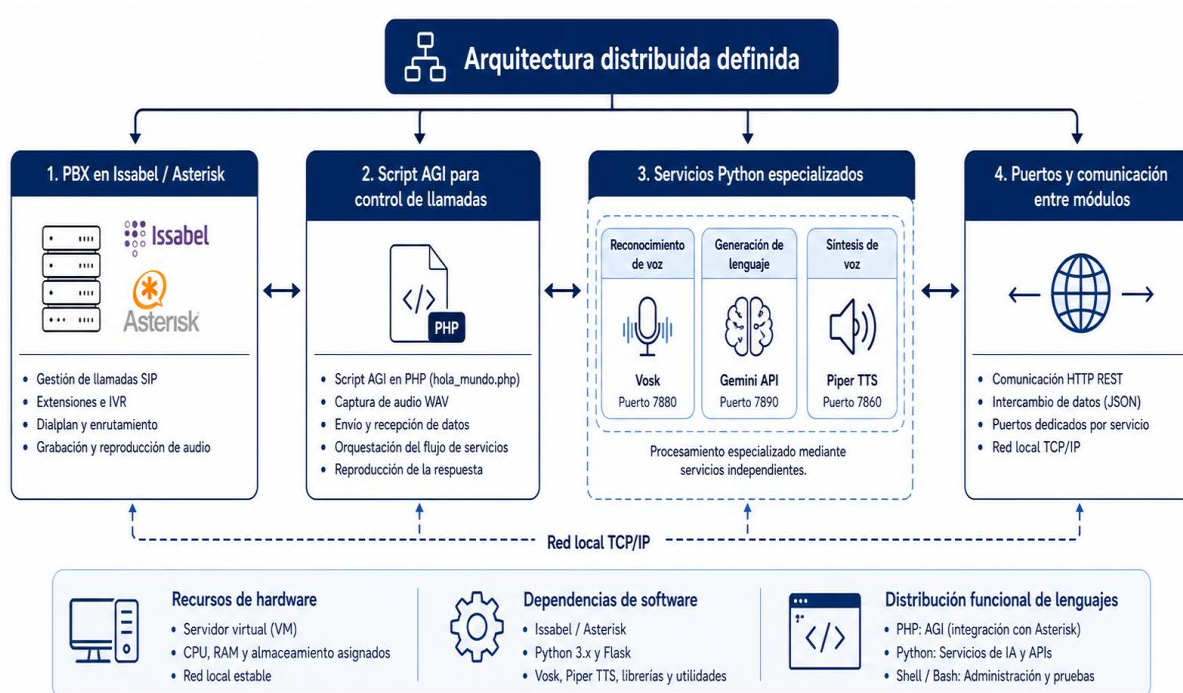
3.12 Síntesis de lo realizado en la fase de diseño

Como resultado de esta fase se presenta un resumen que se detalla en la Figura 4 en donde se definió una arquitectura distribuida compuesta por una PBX en Issabel/Asterisk, un script AGI para el control de llamadas y tres servicios Python especializados en reconocimiento de voz, generación de lenguaje y síntesis. También se determinaron los recursos de hardware, las

dependencias de software, los puertos de comunicación y la distribución funcional de los lenguajes empleados.

Figura 4

Esquema de diseño final



En esta etapa no se ejecutó aún toda la integración final, pero sí se estableció con claridad qué tecnologías serían utilizadas, por qué fueron seleccionadas y qué papel cumpliría cada una dentro de la solución. Esta delimitación fue necesaria para que la implementación desarrollada en el capítulo siguiente no partiera de decisiones improvisadas, sino de un marco técnico previamente fundamentado.

Capítulo IV: Evaluación del prototipo

Los sistemas IVR convencionales suelen depender de menús rígidos basados en tonos DTMF; sin embargo, la incorporación de servicios de reconocimiento de voz y modelos generativos permite transitar hacia interacciones más naturales. En este trabajo se adoptó dicho enfoque con la finalidad de construir un prototipo funcional en el que el usuario pudiera formular una consulta oral y recibir una respuesta sintetizada sin abandonar la sesión telefónica. El diseño respondió a un criterio de factibilidad técnica, reutilización de software disponible y posibilidad de despliegue en un entorno real.

La implementación no se realizó como una instalación sencilla, sino como una secuencia. Primero se habilitó la comunicación SIP en Issabel; después se probó la marcación interna entre extensiones; posteriormente se configuró el IVR base; y finalmente se integraron los servicios de inteligencia artificial hasta obtener una arquitectura completa, operativa y comprobable mediante llamadas reales. Esta estrategia redujo el riesgo de errores cruzados y permitió aislar incidencias de red, formato de audio, rutas de archivos y disponibilidad de modelos.

4.1 Entorno de implementación

La implementación se distribuyó entre dos nodos principales. El primero correspondió al servidor Issabel, ejecutado en una máquina virtual y accesible mediante la dirección IP 192.168.0.104. El segundo correspondió al equipo anfitrión, con dirección IP 192.168.0.102, en el cual se ejecutaron los scripts en Python responsables de la transcripción, la generación de respuestas y la síntesis de voz. Los componentes y recursos que se detalla en la Tabla 5 permitió mantener a la central telefónica concentrada en el manejo de la llamada y delegar el procesamiento a un host distinto.

Tabla 5*Componentes y recursos empleados en la implementación*

Componente	Tecnología	Función	Dato clave
Servidor PBX	Issabel sobre Asterisk 16.7.0	Administración de extensiones, IVR y ejecución del AGI	IP 192.168.0.104
Host de servicios	Windows con Python	Ejecución de STT, Gemini y TTS	IP 192.168.0.102
Transcripción	Vosk	Conversión de voz a texto	Puerto 7880
Generación de respuesta	Gemini API	Producción de respuestas breves en español	Puerto 7890
Síntesis de voz	Piper	Conversión de texto a audio WAV	Puerto 7860
Terminales SIP	Zoiper	Pruebas de marcación y audio	Extensiones 100 y 101

Nota. Los puertos corresponden a la configuración validada durante la integración final del sistema.

La elección de Vosk respondió a su capacidad de reconocimiento de voz fuera de línea y a la disponibilidad de modelos en español, lo que resultó adecuado con recursos controlados (Alpha Cephei, n.d.). Por su parte, Gemini se empleó como motor de generación de respuestas debido a la disponibilidad de una API para integrar respuestas breves en lenguaje natural

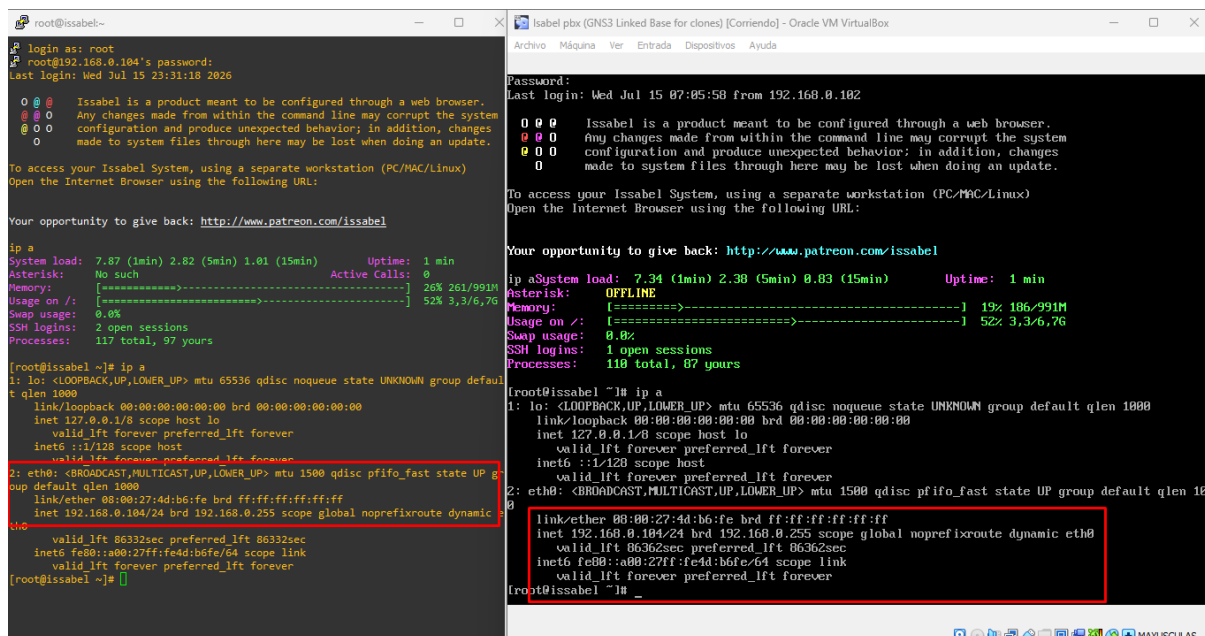
(Google, n.d.). Finalmente, Piper se seleccionó como sintetizador por su funcionamiento local, su bajo costo operativo y la disponibilidad de modelos de voz en formato ONNX (OHF-Voice, n.d.).

4.2 Configuración inicial de Issabel

La primera actividad consistió en verificar el funcionamiento base del servidor Issabel como se representa en la figura 5. Para ello se confirmó su arranque correcto, la disponibilidad del servicio de telefonía y la asignación de una dirección IP accesible dentro de la red local. Una vez obtenida la conectividad, se ingresó a la interfaz web de administración mediante navegador seguro en la dirección del servidor y se comprobó que los módulos del sistema, la base de datos y el servicio web se encontraran en estado operativo.

Figura 5

Comprobación y configuraciones iniciales en servidor Isabell



```

root@issabel:~# login as: root
root@192.168.0.104's password:
Last login: Wed Jul 15 23:31:18 2026

0 0 0 Issabel is a product meant to be configured through a web browser.
0 0 0 Any changes made from within the command line may corrupt the system
0 0 0 configuration and produce unexpected behavior; in addition, changes
0 0 0 made to system files through here may be lost when doing an update.

To access your Issabel System, using a separate workstation (PC/MAC/Linux)
Open the Internet Browser using the following URL:

Your opportunity to give back: http://www.patreon.com/issabel

ip #
System load: 7.87 (1min) 2.82 (5min) 1.01 (15min) Uptime: 1 min
Asterisk: No such Active Calls: 0
Memory: [=====] 26% 261/991M
Swap usage: 0.0%
SSH logins: 2 open sessions
Processes: 117 total, 97 yours

[root@issabel ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:4d:b6:fe brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.104/24 brd 192.168.0.255 scope global noprefixroute dynamic
        valid_lft 86332sec preferred_lft 86332sec
    inet6 fe80::a00:27ff:fe4d:b6fe:64 scope link
        valid_lft forever preferred_lft forever
[root@issabel ~]#

Password:
Last login: Wed Jul 15 07:05:58 from 192.168.0.102

0 0 0 Issabel is a product meant to be configured through a web browser.
0 0 0 Any changes made from within the command line may corrupt the system
0 0 0 configuration and produce unexpected behavior; in addition, changes
0 0 0 made to system files through here may be lost when doing an update.

To access your Issabel System, using a separate workstation (PC/MAC/Linux)
Open the Internet Browser using the following URL:

Your opportunity to give back: http://www.patreon.com/issabel

ip aSystem load: 7.34 (1min) 2.38 (5min) 0.83 (15min) Uptime: 1 min
Asterisk: OFFLINE
Memory: [=====] 19% 186/991M
Swap usage: 0.0%
SSH logins: 1 open sessions
Processes: 110 total, 87 yours

[root@issabel ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:4d:b6:fe brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.104/24 brd 192.168.0.255 scope global noprefixroute dynamic eth0
        valid_lft 86362sec preferred_lft 86362sec
    inet6 fe80::a00:27ff:fe4d:b6fe:64 scope link
        valid_lft forever preferred_lft forever
[root@issabel ~]#
  
```

Esta validación inicial fue indispensable para evitar que fallas de infraestructura afectaran las etapas posteriores. Issabel proporcionó la capa de administración PBX, mientras

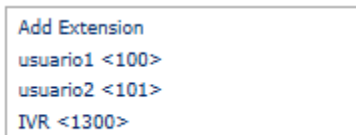
que Asterisk actuó como motor de señalización y audio. (Issabel, n.d.; The Asterisk Project, n.d.).

4.3 Creación de extensiones SIP y validación de llamadas internas

Se procedió a crear dos extensiones SIP dentro del módulo PBX Configuration. Las extensiones asignadas para las pruebas fueron 100 y 101 como se detalla en la figura 6, cada una con sus respectivas credenciales de autenticación. Posteriormente, dichas cuentas se configuraron en dos instancias del softphone Zoiper, lo que permitió simular terminales telefónicas independientes dentro de la misma red. En pruebas complementarias también se contempló la equivalencia de la configuración en un cliente iOS, utilizando como parámetros básicos la dirección del servidor PBX, el usuario SIP, la contraseña y el dominio correspondiente.

Figura 6

Creación de extensiones para realizar pruebas además de un IVR

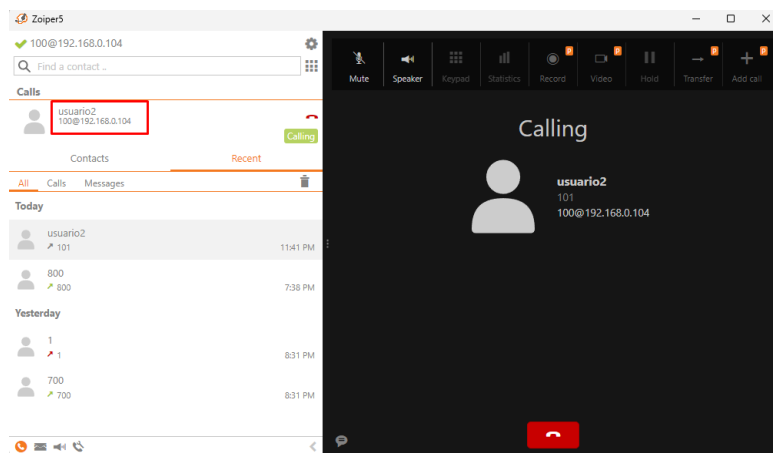


```
Add Extension
usuario1 <100>
usuario2 <101>
IVR <1300>
```

En la figura 7 se realiza la prueba de marcación interna entre ambas extensiones resultó exitosa, lo que confirmó la validez del registro SIP, la negociación de audio y la conectividad entre terminales.

Figura 7

Prueba de marcación entre extensiones



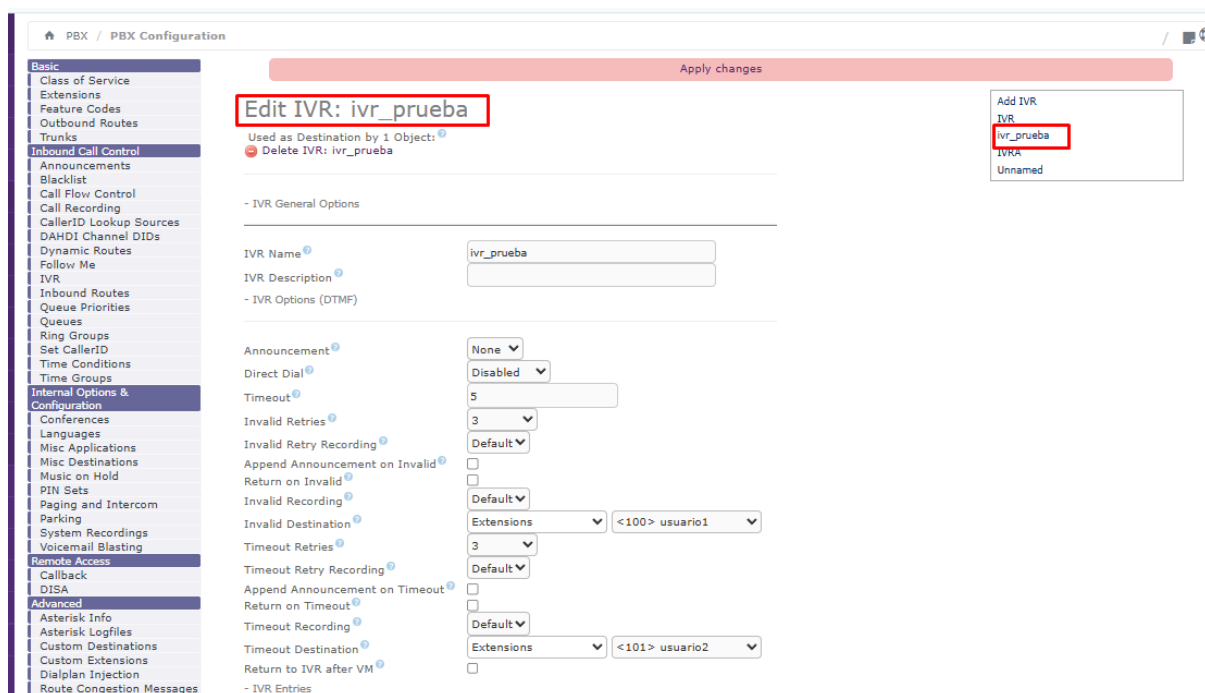
Esta fase fue crítica porque demostró que la infraestructura telefónica funcionaba correctamente antes de añadir cualquier componente de inteligencia artificial.

4.4 Configuración del IVR base en Issabel

Una vez verificada la telefonía interna, se creó un IVR de prueba en Issabel con el propósito de establecer un punto de entrada controlado hacia la lógica conversacional. En la figura 8 se detalla esta etapa, donde se configuraron las opciones generales del IVR, los destinos por tiempo de espera y los comportamientos frente a entradas inválidas. El primer intento de acceso se realizó mediante el código 700; sin embargo, la plataforma reportó que dicho número ya estaba reservado. Por esta razón, se decidió usar el número 800, evitando conflictos con códigos internos preexistentes.

Figura 8

Configuración de IVR



La resolución de este conflicto evidenció una consideración importante en despliegues PBX: no todo número disponible visualmente es realmente libre a nivel funcional, este error permitió validar nuevas rutas de servicio y características predeterminadas del sistema, práctica recomendable cuando se trabaja con centrales.

4.5 Desarrollo del script AGI en PHP

Superada la fase de IVR, se implementó el script AGI encargado de coordinar la interacción entre la llamada y los servicios externos. El script que se detalla en el Anexo 5 se alojó en la ruta `/var/lib/asterisk/agi-bin/` y fue validado antes de su ejecución. El objetivo principal consistió en contestar la llamada, reproducir un mensaje de inicio, generar un archivo temporal de grabación en formato WAV, enviar ese archivo al servicio de transcripción, reenviar el texto obtenido al servicio de Gemini y finalmente solicitar al servicio TTS un audio de respuesta reproducible por Asterisk.

La estructura del AGI se diseñó con una lógica modular. Primero se encapsularon funciones de comunicación HTTP para el envío de JSON y archivos multipart. Luego se

incorporaron procedimientos para reproducir archivos temporales, limpiar recursos y manejar excepciones. Esta decisión simplificó la depuración, ya que cada componente del flujo podía verificarse de manera independiente. Además, el script fue adaptado para almacenar la respuesta sintetizada en el directorio de sonidos de Asterisk y reproducirla mediante `stream_file`, lo que garantizó compatibilidad con el entorno telefónico.

En la figura 9 se muestra un extracto de la versión final en donde se añadió un comportamiento conversacional continuo. En lugar de finalizar la llamada después de una sola consulta, el AGI mantuvo el ciclo de escucha y respuesta hasta detectar la frase de cierre “muchas gracias” o hasta que el usuario colgara la llamada. Esta mejora transformó la solución desde un prototipo de una sola interacción hacia un IVR de diálogo breve, más coherente con un escenario de asistencia automatizada.

Figura 9

Extracto del código de programación del archivo AGI

```

GNU nano 2.3.1 Fichero: /var/lib/asterisk/agi-bin/hola_mundo.php

#!/usr/bin/php -q
<?php
require('/var/lib/asterisk/agi-bin/phpagi.php');

$agi = new AGI();
$agi->answer();

function canalColgado($resultado) {
    return is_array($resultado)
        && isset($resultado['result'])
        && (string)$resultado['result'] === '-1';
}

function transcribirAudio($archivo, $agi) {
    $apiUrl = "http://192.168.0.102:7880/transcribe";
    $curlBin = "/usr/bin/curl";

    $agi->verbose("DEBUG: entrando a transcribirAudio con " . $archivo, 1);

    if (!file_exists($archivo)) {
        return "Error: no existe el archivo";
    }

    if (!file_exists($curlBin)) {
        return "Error: no existe /usr/bin/curl";
    }

    $cmd = $curlBin
        . " --connect-timeout 10 --max-time 40 -sS"
        . " -H " . escapeshellarg("Expect:")
        . " -X POST"
        . " -F " . escapeshellarg("audio=@" . $archivo . ";type=audio/wav")
        . " " . escapeshellarg($apiUrl)
        . " 2>&1";

    $agi->verbose("DEBUG: ejecutando curl shell para transcripcion", 1);

    $output = array();
    $exitCode = 0;
    exec($cmd, $output, $exitCode);
}

```

[251 líneas leídas]

^G Ver ayuda ^O Guardar ^R Leer Fich ^Y Pág Ant ^K CortarTxt ^C Pos actual
 ^X Salir ^J Justificar ^W Buscar ^V Pág Sig ^U PegarTxt ^T Ortografia

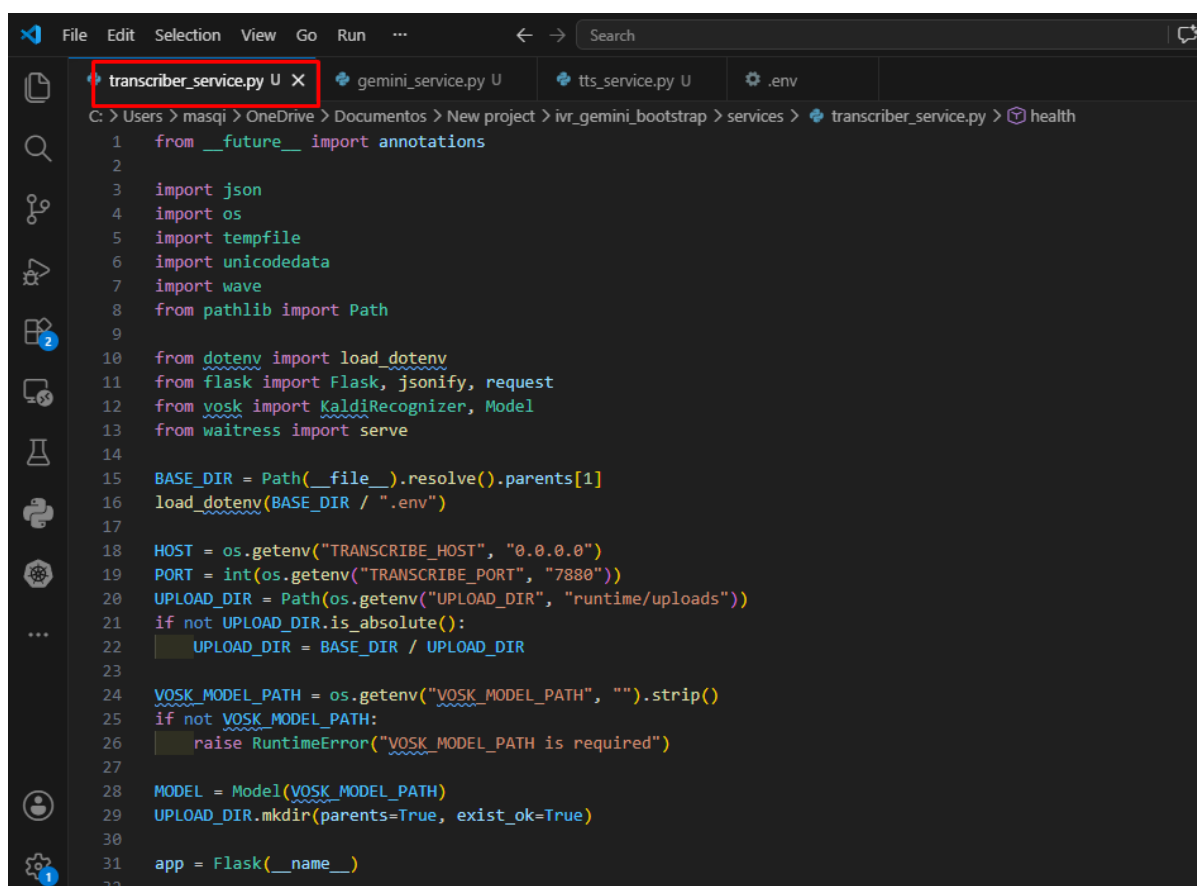
4.6 Implementación del servicio de transcripción con Vosk

En la figura 10 se muestra un extracto del código de programación y en el Anexo 1 el código completo del servicio de transcripción que se desarrolló en Python utilizando Flask y Waitress como servidor HTTP, y Vosk como motor de reconocimiento del habla. Su endpoint principal fue POST /transcribe, encargado de recibir archivos de audio y devolver la cadena transcrita en formato JSON. Durante la implementación se configuró la variable

VOSK_MODEL_PATH para apuntar al modelo vosk-model-small-es-0.42, el cual debió descargarse previamente y ubicarse en una ruta accesible desde el entorno Python. Cuando dicha ruta fue incorrecta o el modelo no existía, el servicio devolvió errores de inicialización, por lo que la validación del directorio resultó obligatoria.

Figura 10

Extracto de código de programación para la transcripción



```

1  from __future__ import annotations
2
3  import json
4  import os
5  import tempfile
6  import unicodedata
7  import wave
8  from pathlib import Path
9
10 from dotenv import load_dotenv
11 from flask import Flask, jsonify, request
12 from vosk import KaldiRecognizer, Model
13 from waitress import serve
14
15 BASE_DIR = Path(__file__).resolve().parents[1]
16 load_dotenv(BASE_DIR / ".env")
17
18 HOST = os.getenv("TRANSCRIBE_HOST", "0.0.0.0")
19 PORT = int(os.getenv("TRANSCRIBE_PORT", "7880"))
20 UPLOAD_DIR = Path(os.getenv("UPLOAD_DIR", "runtime/uploads"))
21 if not UPLOAD_DIR.is_absolute():
22     UPLOAD_DIR = BASE_DIR / UPLOAD_DIR
23
24 VOSK_MODEL_PATH = os.getenv("VOSK_MODEL_PATH", "").strip()
25 if not VOSK_MODEL_PATH:
26     raise RuntimeError("VOSK_MODEL_PATH is required")
27
28 MODEL = Model(VOSK_MODEL_PATH)
29 UPLOAD_DIR.mkdir(parents=True, exist_ok=True)
30
31 app = Flask(__name__)
32

```

Una vez corregida la ruta del modelo, el servicio respondió correctamente tanto desde el equipo anfitrión como desde el servidor Issabel. La verificación se realizó mediante solicitudes al endpoint /health y pruebas de envío de archivos WAV grabados por la propia central. Este comportamiento confirmó que el subsistema de reconocimiento de voz podía operar de manera

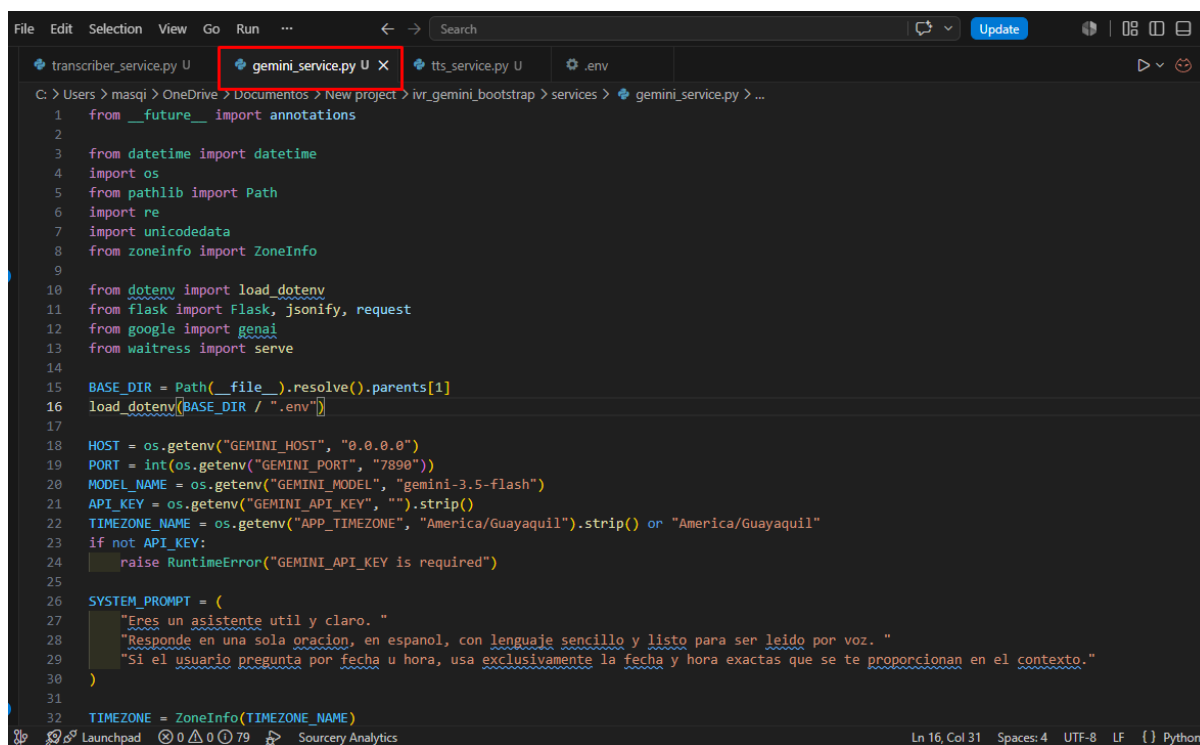
local y estable, sin depender de servicios de terceros para la etapa de transcripción (Alpha Cephei, n.d.).

4.7 Implementación del servicio de generación de respuestas con Gemini

De la misma manera en la figura 11 se muestra un extracto de la programación del servicio de generación de respuestas que se desarrolló en Python con Flask y Waitress y su totalidad se encuentra en el Anexo 2. Su endpoint POST /gemini recibió el texto transcrito por Vosk y devolvió una respuesta breve, en español y con formulación adecuada para lectura por voz. Durante esta etapa se configuró una clave de API y se evaluaron distintos modelos disponibles. Las primeras pruebas generaron errores de cuota agotada y de modelo no disponible para nuevos usuarios, por lo que fue necesario revisar el estado del proyecto en Google AI Studio y ajustar el modelo hasta estabilizar la integración final con gemini-3.5-flash.

Figura 11

Extracto de código de programación para Gemini



```

File Edit Selection View Go Run ... Search Update
transcriber_service.py U gemini_service.py U X tts_service.py U .env
C:\Users\masqi> OneDrive\Documents\New project\ivr_gemini_bootstrap\services> gemini_service.py ...
1 from __future__ import annotations
2
3 from datetime import datetime
4 import os
5 from pathlib import Path
6 import re
7 import unicodedata
8 from zoneinfo import ZoneInfo
9
10 from dotenv import load_dotenv
11 from flask import Flask, jsonify, request
12 from google import generativeai
13 from waitress import serve
14
15 BASE_DIR = Path(__file__).resolve().parents[1]
16 load_dotenv(BASE_DIR / ".env")
17
18 HOST = os.getenv("GEMINI_HOST", "0.0.0.0")
19 PORT = int(os.getenv("GEMINI_PORT", "7890"))
20 MODEL_NAME = os.getenv("GEMINI_MODEL", "gemini-3.5-flash")
21 API_KEY = os.getenv("GEMINI_API_KEY", "").strip()
22 TIMEZONE_NAME = os.getenv("APP_TIMEZONE", "America/Guayaquil").strip() or "America/Guayaquil"
23 if not API_KEY:
24     raise RuntimeError("GEMINI_API_KEY is required")
25
26 SYSTEM_PROMPT = (
27     "Eres un asistente útil y claro. "
28     "Responde en una sola oración, en español, con lenguaje sencillo y listo para ser leído por voz. "
29     "Si el usuario pregunta por fecha u hora, usa exclusivamente la fecha y hora exactas que se te proporcionan en el contexto."
30 )
31
32 TIMEZONE = ZoneInfo(TIMEZONE_NAME)

```

Ln 16, Col 31 Spaces: 4 UTF-8 LF Python

Uno de los problemas funcionales más relevantes surgió cuando el usuario realiza consultas de fecha actual y el sistema respondió con una referencia temporal desactualizada. Para corregirlo no bastó con modificar el prompt; fue necesario incorporar una lógica en el servicio para detectar preguntas relacionadas con fecha y hora, obtener el valor real del sistema y responderlo utilizando la zona horaria America/Guayaquil. Como resultado, el servicio dejó de depender exclusivamente de la inferencia del modelo para ese tipo de consultas y pasó a utilizar datos temporales reales del entorno de ejecución.

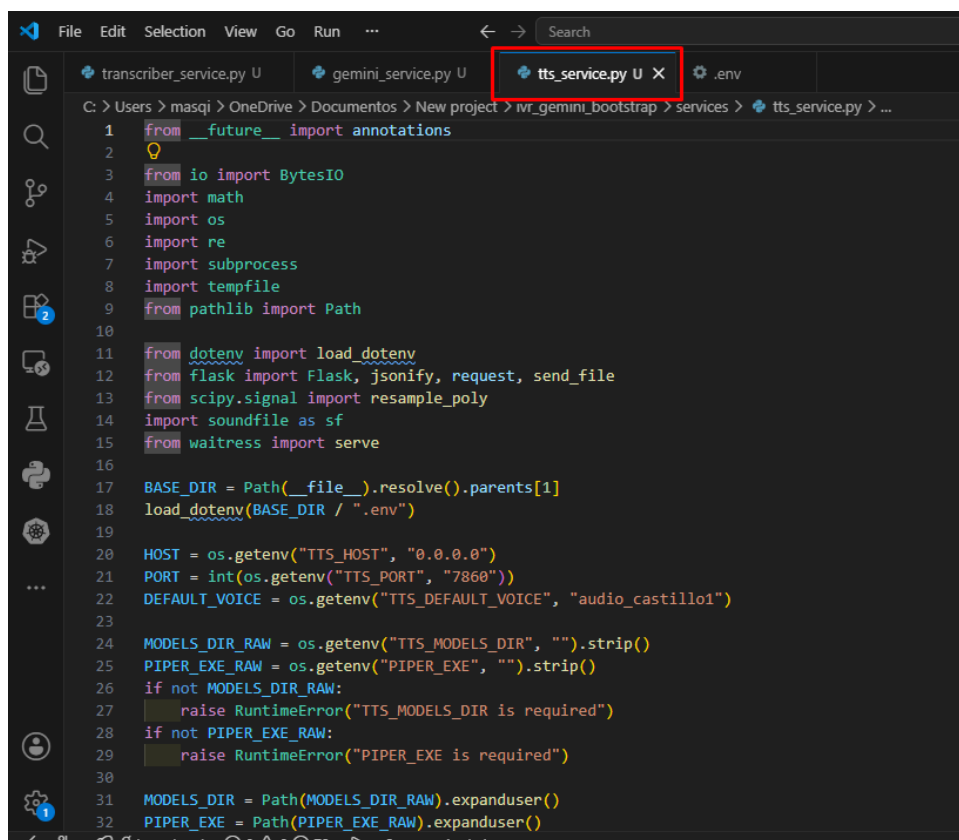
Esta decisión aumentó la confiabilidad del sistema y redujo el riesgo de respuestas erróneas en preguntas sensibles al contexto temporal. Además, el endpoint /health del servicio pasó a informar el modelo activo, el puerto de escucha, la zona horaria configurada y la fecha y hora actuales, lo que facilitó las tareas de diagnóstico en la fase de integración (Google, n.d.).

4.8 Implementación del servicio de síntesis de voz con Piper

La programación se desarrolló mediante Python como se muestra en la Figura 12 y su contenido completo en el Anexo 3, la síntesis de voz se implementó mediante Piper, herramienta instalada en el entorno Python y complementada con un conjunto de voces almacenadas en un directorio específico de modelos ONNX. El servicio expuso el endpoint POST /convert, que recibió una cadena de texto y un alias de voz, generó un archivo WAV temporal y lo transformó a 8 kHz en mono, formato compatible con la reproducción en Asterisk. La normalización de frecuencia de muestreo fue una etapa necesaria, ya que varios motores TTS producen audio con características pensadas para multimedia general, no para telefonía.

Figura 12

Extracto de código de programación del archivo para TTS



```

1 from __future__ import annotations
2
3 from io import BytesIO
4 import math
5 import os
6 import re
7 import subprocess
8 import tempfile
9 from pathlib import Path
10
11 from dotenv import load_dotenv
12 from flask import Flask, jsonify, request, send_file
13 from scipy.signal import resample_poly
14 import soundfile as sf
15 from waitress import serve
16
17 BASE_DIR = Path(__file__).resolve().parents[1]
18 load_dotenv(BASE_DIR / ".env")
19
20 HOST = os.getenv("TTS_HOST", "0.0.0.0")
21 PORT = int(os.getenv("TTS_PORT", "7860"))
22 DEFAULT_VOICE = os.getenv("TTS_DEFAULT_VOICE", "audio_castillo1")
23
24 MODELS_DIR_RAW = os.getenv("TTS_MODELS_DIR", "").strip()
25 PIPER_EXE_RAW = os.getenv("PIPER_EXE", "").strip()
26 if not MODELS_DIR_RAW:
27     raise RuntimeError("TTS_MODELS_DIR is required")
28 if not PIPER_EXE_RAW:
29     raise RuntimeError("PIPER_EXE is required")
30
31 MODELS_DIR = Path(MODELS_DIR_RAW).expanduser()
32 PIPER_EXE = Path(PIPER_EXE_RAW).expanduser()
  
```

Durante las primeras pruebas apareció una excepción de permisos en Windows porque el archivo temporal de salida permanecía en uso al momento de intentar eliminarlo. La corrección consistió en leer el contenido binario del archivo, enviarlo desde memoria mediante BytesIO y eliminar la versión temporal inmediatamente después. Tras este ajuste, el endpoint respondió con código HTTP 200, tipo MIME audio/wav y un archivo reproducible desde Issabel, lo que confirmó la operatividad completa del subsistema TTS (OHF-Voice, n.d.).

4.9 Integración de la arquitectura completa

En la figura 13 se muestra la integración que se llevó a cabo. En primer lugar, cada servicio se validó individualmente con su correspondiente endpoint /health. En segundo lugar, se verificó la conectividad desde Issabel hacia el host de servicios usando solicitudes HTTP

directas. En tercer lugar, se ejecutaron pruebas parciales: transcripción de archivos ya grabados, consultas manuales al servicio Gemini y generación de audio por TTS. Solo después de confirmar el funcionamiento aislado de cada módulo se procedió a enlazarlos desde el script AGI.

Figura 13

Implementación del proyecto en conjunto

```

Windows PowerShell
LOG (VoskAPI:ReadDataFiles():model.cc:248) Loading i-vector extractor from C:/Users/masqi/OneDrive/Escritorio/Trabajo de Titulacion II/Capitulo 3/vosk-model-small-es-0.42/vosk-model-small-es-0.42/ivector/final.ie
LOG (VoskAPI:ComputeDerivedVars():ivector-extractor.cc:183) Computing derived variables for iVector extractor
LOG (VoskAPI:ComputeDerivedVars():ivector-extractor.cc:204) Done.
LOG (VoskAPI:ReadDataFiles():model.cc:282) Loading HCL and G from C:/Users/masqi/OneDrive/Escritorio/Trabajo de Titulacion II/Capitulo 3/vosk-model-small-es-0.42/vosk-model-small-es-0.42/graph/HCLr.fst C:/Users/masqi/OneDrive/Escritorio/Trabajo de Titulacion II/Capitulo 3/vosk-model-small-es-0.42/vosk-model-small-es-0.42/graph/Gr.fst
LOG (VoskAPI:ReadDataFiles():model.cc:308) Loading winfo C:/Users/masqi/OneDrive/Escritorio/Trabajo de Titulacion II/Capitulo 3/vosk-model-small-es-0.42/vosk-model-small-es-0.42/graph/phones/word_boundary.int

root@issabel:~# asterisk -rvvv
Asterisk 16.7.0, Copyright (C) 1999 - 2018, Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for details.
This is free software, with components licensed under the GNU General Public
License version 2 and other licenses; you are welcome to redistribute it under
certain conditions. Type 'core show license' for details.
-----
Connected to Asterisk 16.7.0 currently running on issabel (pid = 2333)
[2026-07-16 00:12:06] NOTICE[2633]: chan_sip.c:15985 sip_reg_timeout: -- Registration for '
otavalo@10.0.60.16' timed out, trying again (Attempt #76)
-- Using SIP RTP TOS bits 184
-- Using SIP RTP CoS mark 5
Executing [800@from-internal:1] NoOp("SIP/100-00000002", "Prueba AGI hola_mundo") in new
stack
- Executing [800@from-internal:2] Answer("SIP/100-00000002", "") in new stack
- Executing [800@from-internal:3] AGI("SIP/100-00000002", "hola_mundo.php") in new stack
Launched AGI Script /var/lib/asterisk/agi-bin/hola_mundo.php
[2026-07-16 00:12:28] NOTICE[2633]: chan_sip.c:15985 sip_reg_timeout: -- Registration for '
otavalo@10.0.60.16' timed out, trying again (Attempt #76)
issabel*CLI>

PS C:\Users\masqi\OneDrive\Documentos\New project\ivr_gemini_bootstrap>
cd "C:\Users\masqi\OneDrive\Documentos\New project\ivr_gemini_bootstrap"

PS C:\Users\masqi\OneDrive\Documentos\New project\ivr_gemini_bootstrap>py
services\gemini_service.py
PS C:\Users\masqi\OneDrive\Documentos\New project\ivr_gemini_bootstrap>py
services\gemini_service.py
PS C:\Users\masqi\OneDrive\Documentos\New project\ivr_gemini_bootstrap>py
services\gemini_service.py
PS C:\Users\masqi\OneDrive\Documentos\New project\ivr_gemini_bootstrap>c
d "C:\Users\masqi\OneDrive\Documentos\New project\ivr_gemini_bootstrap"
PS C:\Users\masqi\OneDrive\Documentos\New project\ivr_gemini_bootstrap>py
services\gemini_service.py

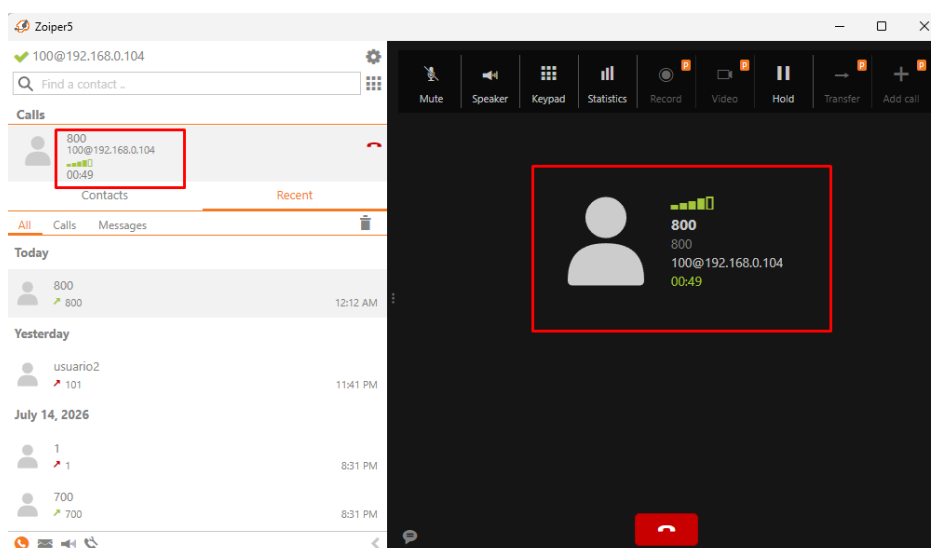
a6y7.wav'
PS C:\Users\masqi\OneDrive\Documentos\New project\ivr_gemini_bootstrap>
cd "C:\Users\masqi\OneDrive\Documentos\New project\ivr_gemini_bootstrap"
PS C:\Users\masqi\OneDrive\Documentos\New project\ivr_gemini_bootstrap>py
services\tts_service.py
PS C:\Users\masqi\OneDrive\Documentos\New project\ivr_gemini_bootstrap>c
d "C:\Users\masqi\OneDrive\Documentos\New project\ivr_gemini_bootstrap"
PS C:\Users\masqi\OneDrive\Documentos\New project\ivr_gemini_bootstrap>py
services\tts_service.py
PS C:\Users\masqi\OneDrive\Documentos\New project\ivr_gemini_bootstrap>py
services\tts_service.py
PS C:\Users\masqi\OneDrive\Documentos\New project\ivr_gemini_bootstrap>py
services\tts_service.py

```

En la Figura 14 se realizó la prueba integral final, el usuario llamó al número 800, emitió una consulta, el sistema generó la transcripción, formuló una respuesta con Gemini, obtuvo el WAV sintetizado y reprodujo el resultado dentro de la misma llamada. Las bitácoras de Asterisk mostraron el registro del archivo de entrada, la transcripción detectada, la respuesta del modelo y el estado exitoso del TTS, evidenciando que la cadena completa de procesamiento funcionó de extremo a extremo.

Figura 14

Comprobación de funcionamiento realizando llamada al número 800



4.10 Ajustes y resolución de incidencias

El desarrollo no estuvo exento de incidencias técnicas. La primera fue el conflicto del número 700, reservado por una función interna, lo que obligó a reasignar el acceso del IVR a 800. La segunda fue la ausencia inicial del modelo Vosk o su referencia incorrecta dentro del archivo .env, el mismo que se encuentra en el Anexo 4, situación que impedía arrancar el servicio de transcripción. La tercera estuvo asociada a problemas de cuota y compatibilidad de modelos en Gemini, los cuales se resolvieron mediante la revisión del proyecto activo y la selección de un modelo compatible con el entorno disponible.

A nivel de integración de red también se identificaron tiempos de espera al transcribir desde el AGI. Estos fallos se originaron, en distintos momentos, por servicios no activos, pruebas realizadas contra hosts equivocados o diferencias entre llamadas locales y remotas. La solución consistió en validar cada endpoint desde ambas máquinas, verificar el direccionamiento IP real y mantener los servicios Python ejecutándose antes de iniciar las llamadas de prueba.

Otro ajuste importante se produjo en la etapa TTS. Aunque Piper ya generaba audio correctamente en pruebas locales, el servidor devolvía error interno cuando el archivo temporal se intentaba borrar mientras todavía era procesado por Flask. El envío del audio desde memoria resolvió el bloqueo. Finalmente, se incorporó un criterio de cierre conversacional controlado por la frase “muchas gracias”, lo cual mejoró la experiencia de uso y evitó la terminación prematura de la sesión telefónica.

4.11 Arquitectura final implementada

La arquitectura final se muestra en la Tabla 6 en donde quedó conformada por un núcleo PBX en Issabel/Asterisk y tres servicios especializados que operan como capa de inteligencia. El AGI asumió el rol de orquestador transaccional, mientras que Vosk, Gemini y Piper resolvieron respectivamente la transcripción, la generación semántica y la síntesis de voz. Esta distribución permitió mantener la lógica telefónica cerca de la central y el procesamiento especializado en servicios HTTP independientes, con la ventaja adicional de poder sustituir o escalar módulos sin rediseñar toda la solución.

Tabla 6

Flujo funcional de la arquitectura conversacional implementada

Etapa	Entrada	Proceso	Salida
1	Llamada SIP a 800	Issabel enruta la llamada al AGI	Sesión de voz activa
2	Mensaje del usuario	Asterisk graba audio WAV temporal	Archivo de audio
3	Archivo WAV	Vosk transcribe mediante /transcribe	Texto de consulta

4	Texto transcrito	Gemini genera una respuesta breve	Texto de respuesta
5	Texto de respuesta	Piper sintetiza audio por /convert	WAV a 8 kHz
6	WAV final	Asterisk reproduce audio al usuario	Continuidad o cierre

La principal fortaleza de la arquitectura radica en su modularidad. Cada servicio puede probarse, reiniciarse o modificarse sin afectar directamente a los demás, siempre que se preserve el contrato de entrada y salida definido por HTTP y JSON. Esta propiedad es particularmente valiosa en sistemas experimentales, donde los modelos, los tiempos de respuesta y los formatos de audio suelen requerir ajustes frecuentes.

4.12 Evaluación del asistente conversacional implementado sobre Issabel

La evaluación experimental del prototipo se realizó con el fin de medir el comportamiento del asistente conversacional integrado a la central telefónica Issabel. Para lo cual se formuló diez preguntas básicas y, utilizó la expresión de salida "muchas gracias" para cerrar la llamada. La evidencia obtenida permitió calcular indicadores simples de precisión de transcripción, exactitud de respuesta, tiempo de procesamiento, tiempo total percibido, tasa de automatización y eficiencia operativa.

4.12.1 Metodología de evaluación

La metodología se basó en la observación de una interacción real registrada por Asterisk, el script AGI y los servicios externos de reconocimiento de voz, generación de respuesta y síntesis de voz. Para lo cual se seleccionó 10 preguntas que se detalla en la Tabla 7 y su procesamiento se muestra en los Anexos 6, cada pregunta fue contrastada con tres elementos: la

transcripción detectada por el servicio de voz a texto, la respuesta generada por el modelo conversacional y el tiempo estimado requerido para completar el ciclo de respuesta. A partir de esta información se construyeron indicadores simples y directamente interpretables.

Para el cálculo de tiempos se utilizaron los identificadores numéricos incluidos en los nombres de los archivos temporales de audio y de salida TTS. El tiempo de procesamiento de respuesta (TPR) se calculó como la diferencia entre la marca temporal del audio capturado y la marca temporal del archivo sintetizado. El tiempo total de respuesta percibida (TRP) se calculó como la diferencia entre la marca temporal del audio capturado y el inicio de la siguiente ventana de escucha. Cuando una consulta requirió repetición, como ocurrió con la pregunta sobre internet, el tiempo se consolidó hasta obtener una respuesta semánticamente correcta que se detallan en la Tabla 8.

Tabla 7

Preguntas básicas empleadas para la evaluación funcional del asistente

N	Pregunta planificada
1	Hola.
2	¿Cómo estás?
3	¿Qué hora es?
4	¿Qué día es hoy?
5	¿Cuánto es dos más dos?
6	¿De qué color es el sol?
7	¿Cuál es tu nombre?
8	¿Qué es internet?
9	¿Qué es la fibra óptica?

4.12.2 Resultados obtenidos

Los resultados muestran que el sistema logró completar la totalidad de las consultas.

Tabla 8*Resultados detallados por pregunta, transcripción y tiempos estimados*

N	Pregunta	Transcripción detectada	Éxito STT	Evaluación de respuesta	TPR (s)	TRP (s)	Observación
1	Hola.	hola	Sí, exacta	Correcta	23	25	Interacción inicial correcta.
2	¿Cómo estás?	cómo estás	Sí, exacta	Correcta	23	29	Respuesta coherente y contextual.
3	¿Qué hora es?	hora	Sí, útil	Incorrecta	21	26	La respuesta no coincide con la hora del registro de Asterisk del 16 de julio de 2026.
4	¿Qué día es hoy?	día hoy	Sí, útil	Incorrecta	24	30	La fecha correcta del registro era el 16 de julio de 2026.
5	¿Cuánto es dos más dos?	cuánto es dos más dos	Sí, exacta	Correcta	19	21	La consulta matemática fue resuelta sin errores.
6	¿De qué color es el sol?	de qué color el sol	Sí, útil	Parcial	20	25	La respuesta es científicamente válida, pero excede la simplicidad esperada para la prueba.

7	¿Cuál es tu nombre?	cuál es tu nombre	Sí, exacta	Correcta	17	22	La identidad funcional del asistente se comunica adecuadamente.
8	¿Qué es internet?	Intento 1: es de internet. Intento 2: qué es internet.	Sí, con repetición	Correcta con repetición	49	58	La consulta requirió una repetición para alcanzar la respuesta esperada.
9	¿Qué es la fibra óptica?	el fibra óptica	Sí, útil	Correcta	23	33	La respuesta describe de forma clara el concepto solicitado.
10	¿Qué es una señal digital?	es una señal digital	Sí, útil	Correcta	23	32	La definición fue aceptable para un usuario general.

TPR es el tiempo de procesamiento de respuesta estimado entre el archivo de audio grabado y la generación del archivo TTS.

TRP es el tiempo total percibido hasta la siguiente ventana de escucha. Ambos valores se estimaron a partir de la secuencia temporal.

Para determinar los indicadores de precisión, automatización y eficiencia se tomo en cuenta la cantidad de numero total de pruebas por el porcentaje de éxito en cada una de los parámetros que se ha seleccionado esto viene indicado en la Tabla 9 con sus respectivos valores y su interpretación.

Tabla 9

Indicadores consolidados de precisión, automatización y eficiencia

Indicador	Valor	Interpretación
Consultas planificadas	10	Se evaluaron diez preguntas básicas y una frase final de cierre.
Transcripciones exitosas al primer intento	9/10 (90 %)	Solo la consulta sobre internet requirió repetición.
Transcripciones exactas	9/10 (90 %)	Coincidencia textual alta en cuatro preguntas.
Éxito final de transcripción	10/10 (100 %)	Todas las consultas fueron finalmente comprendidas.
Respuestas correctas	8/10 (80 %)	Ocho respuestas coincidieron con lo esperado sin observaciones.
Respuestas parciales	1/10 (10 %)	La respuesta sobre el color del sol fue útil, pero no tan simple como se esperaba.
Respuestas incorrectas	1/10 (10 %)	Las respuestas sobre fecha y hora presentaron desfase temporal.

Precisión útil de respuesta	10/10 (100 %)	Incluye respuestas correctas y parciales consideradas útiles.
Índice ponderado de respuesta	85 %	Se asignó valor 1 a correctas, 0.5 a parciales y 0 a incorrectas.
Tasa de automatización bruta	10/10 (100 %)	Ninguna consulta requirió intervención humana.
Tasa de automatización efectiva	8/10 (80 %)	Ocho consultas terminaron en respuestas útiles para el usuario.
TPR promedio	24.2 s	Tiempo de procesamiento estimado desde el audio grabado hasta el archivo TTS.
TPR mínimo	17 s	Consulta 7.
TPR máximo	49 s	Consulta 8, con repetición.
TRP promedio	30.1 s	Tiempo total percibido hasta la siguiente ventana de escucha.
TRP mínimo	21 s	Consulta 5.
TRP máximo	58 s	Consulta 8, con repetición.
Eficiencia operativa total	1.99 consultas/min	Diez consultas en 301 segundos de interacción.

Eficiencia operativa útil	1.59 consultas útiles/min	Ocho respuestas útiles en 301 segundos.
---------------------------	---------------------------	---

4.12.3 Análisis de indicadores

La precisión de transcripción al primer intento alcanzó el 90 %, ya que nueve de las diez preguntas fueron comprendidas en una sola emisión. Si se considera la comprensión final del mensaje, el reconocimiento de voz llegó al 100 %, debido a que la única consulta inicialmente ambigua pudo resolverse mediante repetición. Sin embargo, la precisión exacta de la transcripción fue de 90 %, lo cual evidencia que el sistema siempre reproduce literalmente la pregunta del usuario, en la mayoría de los casos logra conservar el sentido semántico suficiente para continuar el flujo conversacional.

En cuanto a la calidad de las respuestas, el sistema obtuvo ocho respuestas correctas, una parcial y una incorrectas. Esto equivale a una precisión estricta de 90 % y a una precisión útil de 100 %. El índice ponderado de respuesta fue de 85 %, valor que resume de manera equilibrada la calidad del desempeño al asignar peso completo a las respuestas correctas, peso intermedio a las respuestas parciales y valor nulo a las respuestas incorrectas. Estos resultados indican que el prototipo ya es funcional para consultas sencillas

Respecto del rendimiento temporal, el TPR promedio fue de 24.2 segundos y el TRP promedio fue de 30.1 segundos. El mejor escenario correspondió a la consulta "¿Cuál es tu nombre?", con 17 segundos de procesamiento y 22 segundos de tiempo total percibido. El peor escenario fue la pregunta "¿Qué es internet?", cuyo ciclo se extendió a 49 segundos de procesamiento y 58 segundos de tiempo total debido a la necesidad de repetir la pregunta. En consecuencia, la latencia del sistema se mantiene dentro de márgenes aceptables, pero todavía es elevada para un entorno productivo con alta exigencia de inmediatez.

La tasa de automatización bruta fue de 100 %, puesto que las diez consultas fueron atendidas enteramente por la arquitectura implementada. Cuando se considera solo la automatización efectiva, es decir, las respuestas útiles para el usuario, el valor desciende a 90 %. Finalmente, la eficiencia operativa total fue de 1.99 consultas por minuto, mientras que la eficiencia operativa útil fue de 1.59 consultas útiles por minuto. Estas cifras muestran que el sistema ya es capaz de sostener un diálogo autónomo, aunque el costo temporal por respuesta sigue siendo el principal factor limitante.

4.12.4 Conclusión de la evaluación

La evidencia experimental permite afirmar que el asistente conversacional desarrollado sobre Issabel, Vosk, Gemini y Piper funciona de manera integral y automatizada para consultas básicas formuladas por voz. El sistema completo logró reconocer, procesar y responder diez consultas sin asistencia humana, con una precisión útil de respuesta del 90 % y una comprensión final del 100 %.

4.13 Resultados de validación

Los resultados obtenidos permitieron verificar el cumplimiento del objetivo técnico del capítulo. En primer lugar, se comprobó la operación correcta de las extensiones SIP y de la marcación interna. En segundo lugar, el IVR logró invocar el AGI desde el número 800 sin conflictos de enrutamiento. En tercer lugar, la central generó y almacenó correctamente los archivos de audio temporales, los cuales fueron interpretados por el servicio de transcripción. En cuarto lugar, Gemini produjo respuestas breves y coherentes en español. Finalmente, Piper sintetizó el contenido de salida en un formato reproducible por Asterisk, cerrando el ciclo conversacional.

Adicionalmente, la versión final del sistema resolvió dos requisitos de calidad relevantes: la consulta por fecha y hora se respondió con datos reales del sistema, y la llamada dejó de finalizar tras una sola pregunta. En su lugar, la conversación permaneció

activa hasta que el usuario pronunciaba “muchas gracias” o decidía colgar. Con ello, la arquitectura alcanzó un nivel de interacción esperado en un IVR.

5. Conclusiones

El proceso descrito demuestra que es técnicamente viable integrar una central Issabel con servicios modernos de inteligencia artificial mediante un esquema híbrido basado en Asterisk AGI y microservicios HTTP. La solución construida no solo permitió automatizar la captura, comprensión y respuesta de consultas de voz, sino que también dejó una base reproducible para nuevas funciones, entre ellas consultas sobre fuentes de datos estructuradas, historiales de interacción o reglas de negocio específicas.

Asimismo, la experiencia de implementación evidenció que la validación incremental es una estrategia adecuada para este tipo de proyectos, ya que facilita la detección temprana de fallas relacionadas con puertos, modelos, rutas de archivos, formatos de audio y disponibilidad de servicios externos. Como línea futura de mejora se recomienda reducir la latencia total, incorporar memoria contextual de diálogo y enlazar el motor conversacional con bases de datos institucionales o archivos tabulares para ofrecer respuestas más precisas y orientadas al dominio de aplicación.

El análisis de requerimientos y la selección tecnológica permitieron construir una base coherente para el desarrollo del sistema IVR conversacional. Cada decisión respondió a una función concreta dentro de la arquitectura: Issabel y Asterisk para la gestión y ejecución telefónica, PHP para el puente con la llamada, Python para los servicios de inteligencia, Vosk para la transcripción, Gemini para la respuesta textual y Piper para la síntesis de voz. De este modo, el capítulo estableció la lógica de diseño sobre la cual se sustenta la fase de implementación.

Los resultados obtenidos durante la fase experimental evidenciaron que el sistema alcanzó un funcionamiento satisfactorio dentro del entorno de prueba, logrando automatizar la totalidad de las consultas planteadas y mantener la continuidad del flujo conversacional entre los distintos módulos implementados. Asimismo, los indicadores de desempeño mostraron niveles favorables de precisión útil y automatización, confirmando que la propuesta cumple con el objetivo de responder consultas básicas de manera autónoma, sin requerir intervención humana en el proceso.

El estudio también permitió identificar limitaciones importantes, principalmente relacionadas con el tiempo de respuesta y la dependencia de servicios externos para la generación de contenido conversacional. Si bien estas restricciones no impidieron la validación del prototipo, sí evidencian la necesidad de optimizar aspectos asociados a la latencia, la disponibilidad de los modelos y la estabilidad general del sistema para su eventual aplicación en contextos más exigentes o de producción real.

La articulación de manera efectiva entre tecnologías pertenecientes a distintos ámbitos, como la telefonía IP, el reconocimiento de voz, el procesamiento de lenguaje natural y la síntesis de voz, en una sola plataforma operativa. Esta integración demostró que es posible aprovechar herramientas de software libre y servicios de inteligencia artificial para desarrollar soluciones innovadoras de atención automatizada, con una estructura flexible y con potencial de adaptación a escenarios reales de servicio.

6. Recomendaciones

Se recomienda continuar con el perfeccionamiento de las características técnicas y funcionales del sistema con el propósito de mejorar su rendimiento integral. Si bien la arquitectura implementada demostró ser funcional y capaz de establecer comunicación entre los distintos módulos, aún existen aspectos susceptibles de optimización, especialmente en lo relacionado con la coordinación entre los servicios de transcripción, generación de respuestas

y síntesis de voz. Una mejora en estos componentes permitiría incrementar la estabilidad operativa, reducir errores durante la interacción y fortalecer la calidad global del servicio ofrecido al usuario.

Se recomienda desarrollar un proceso de evaluación más amplio y sistemático, incorporando un mayor número de pruebas, usuarios y escenarios de interacción. Esto permitiría obtener resultados más representativos en relación con variables como el tiempo de respuesta, la precisión de la transcripción, la coherencia de las respuestas generadas y la estabilidad del sistema frente a distintos tipos de consultas. Asimismo, una muestra más amplia facilitaría la identificación de patrones de comportamiento y contribuiría a establecer métricas más sólidas para valorar el desempeño real de la solución en contextos de uso más cercanos a una implementación práctica.

Se recomienda trabajar específicamente en la disminución del tiempo de retardo existente entre la consulta realizada por el usuario y la respuesta emitida por el sistema. Durante las pruebas efectuadas se observó que, aunque el sistema respondía de manera adecuada, el tiempo de espera entre una intervención y otra aún puede afectar la fluidez natural de la conversación. Reducir este retardo contribuiría significativamente a mejorar la experiencia del usuario, haciendo que la interacción sea más dinámica, cómoda y semejante a una conversación humana en tiempo real. Esta optimización podría abordarse mediante ajustes en el procesamiento de audio, el uso de modelos más ligeros o mejoras en la infraestructura de ejecución.

Se recomienda mantener y profundizar el enfoque modular adoptado en la arquitectura del sistema, debido a que esta estructura facilita la administración, el mantenimiento y la escalabilidad de la solución. La separación de funciones en servicios independientes para reconocimiento de voz, procesamiento del lenguaje natural y síntesis de voz constituye una decisión acertada que permite realizar mejoras puntuales sin comprometer el funcionamiento

total del sistema. En futuras implementaciones, este enfoque podría aprovecharse para incorporar nuevos módulos, integrar bases de datos, añadir mecanismos de autenticación o ampliar las capacidades conversacionales del asistente sin necesidad de rediseñar la arquitectura desde sus cimientos.

Se recomienda conservar el uso de herramientas y plataformas basadas en software libre, debido a las ventajas técnicas, económicas y académicas que estas ofrecen. La utilización de soluciones abiertas como Issabel, Asterisk, Python, Vosk y Piper permitió desarrollar una propuesta funcional con costos considerablemente reducidos en comparación con alternativas comerciales. Además, este tipo de herramientas brinda mayor flexibilidad para personalizar la solución, adaptar los componentes a necesidades específicas y fomentar la replicabilidad del trabajo en futuros proyectos de investigación o desarrollo tecnológico. Se recomienda que futuras investigaciones orienten sus esfuerzos hacia la incorporación de nuevas capacidades funcionales en el asistente conversacional. Entre estas pueden considerarse la gestión de contextos conversacionales más complejos, la conexión con bases de datos institucionales, la personalización de respuestas, la ampliación del vocabulario reconocido y la mejora de la naturalidad en la voz sintetizada. Estas extensiones no solo permitirían elevar el nivel de sofisticación del sistema, sino también acercarlo a escenarios reales de aplicación en los que un asistente de voz pueda brindar soporte automatizado de manera más robusta, útil y eficiente.

Finalmente, se recomienda realizar pruebas en entornos de operación más cercanos a condiciones reales, considerando distintos tipos de red, variaciones en la calidad del audio, diferentes perfiles de usuarios y posibles interrupciones en la comunicación. Este tipo de evaluación permitiría determinar con mayor precisión la capacidad de respuesta del sistema fuera del entorno controlado de laboratorio y contribuiría a identificar limitaciones que no

siempre son visibles en pruebas preliminares. De este modo, sería posible fortalecer la fiabilidad de la propuesta y valorar su factibilidad de adopción en aplicaciones prácticas.

7. REFERENCIAS

- Alphacephei. (2023). Vosk Speech Recognition Toolkit.
<https://alphacephei.com/vosk/>
- Amazon Web Services. (2024). ¿Qué es el NLP? Explicación del procesamiento de lenguaje natural. <https://aws.amazon.com/es/what-is/nlp/>
- Amazon Web Services. (2024). ¿Qué es un LLM? Explicación de los modelos de lenguaje grandes. <https://aws.amazon.com/es/what-is/large-language-model/>
- Ben Ammar, M., & Ben Ammar, O. M. (2021). AI-based automation in customer service: Improving response time and operational efficiency. *Journal of Service Science and Management*, 14(3), 312–329.
- Carrión Vega, A. D., Cuero Ayovi, B. A., & Valarezo Pardo, M. R. (2025). Sistema web integral para un call center usando tecnologías de IA. *Pentaciencias*.
<https://editorialalema.org/index.php/pentaciencias/article/download/1501/2000>
- Celi-Parraga, R. J., Varela-Tapia, E. A., Acosta-Guzmán, I. L., & Montaña-Pulzara, N. (2021). Técnicas de procesamiento de lenguaje natural en la inteligencia artificial conversacional textual. *Alfa Publicaciones*, 3(4.1), 40–52.
- (2025). Propuesta para la implementación de un asistente conversacional con inteligencia artificial integrado al Issabel Call Center para la resolución automática de consultas básicas [Anteproyecto de trabajo de grado]. Universidad Técnica del Norte.
- Fúquene Ardila, H. J. (2024). Procesamiento de lenguaje natural, los transformers y los bots conversacionales. *XIKUA Boletín Científico de la Escuela Superior de Tlahuelilpan*, 12(Especial), 151–160. <https://doi.org/10.29057/xikua.v12iEspecial.12904>

Garzón-Quiroz, M., Del Campo-Saltos, G., & Loo-Ávila, B. (2025). Análisis sistemático sobre la eficiencia comunicativa entre chatbots basados en reglas y modelos de lenguaje natural. *Universitas-XXI, Revista de Ciencias Sociales y Humanas*.

<https://universitas.ups.edu.ec/index.php/universitas/article/view/9020>

Giraldo Forero, A. F., & Orozco Duque, A. F. (2023). Evolución del procesamiento natural del lenguaje. *Tecnológicas*, 26(56). <https://doi.org/10.22430/22565337.2687>

IBM. (2024). ¿Qué es Google Gemini? <https://www.ibm.com/mx-es/think/topics/google-gemini>

Issabel. (2024). Beyond the GUI: The power of AMI, AGI, and ARI in Issabel. <https://www.issabel.com/en/beyond-the-gui-the-power-of-ami-agi-and-ari-in-issabel/>

Issabel. (2025a). Contact Center. <https://www.issabel.com/contact-center-es/>

Issabel. (2025b). IA en el Contact Center de Issabel. <https://www.issabel.com/ia-en-el-contact-center-de-issabel/>

Ministerio de Telecomunicaciones y de la Sociedad de la Información del Ecuador. (2025). Gobierno nacional impulsa la transformación digital con el programa EmpoderaTech Ecuador. <https://www.telecomunicaciones.gob.ec/gobierno-nacional-impulsa-la-transformacion-digital-con-el-programa-empoderatech-ecuador/>

Muñoz Guillena, R. (2024). Procesamiento del lenguaje natural como eje central de la inteligencia artificial [Lección inaugural del Curso Académico 2024-2025]. Universidad de La Rioja.

Netelip. (2021). Centralita virtualizada Issabel + Call Center. <https://www.netelip.com/ar/centralita-virtualizada-modulo-call-center-issabel/>

NTT DATA. (s.f.). Inteligencia artificial y el futuro de los contact centers. <https://es.nttdata.com/insights/blog/contact-centers-futuro-ia>

Valarezo Pardo, M. R., & Cárdenas Villavicencio, O. E. (2025). Comprehensive web system for a call center using AI. PENTACIENCIAS.

Wise Guy Reports. (2024). Mercado de IA para centros de llamadas: tendencias y oportunidades para 2032. <https://www.wiseguyreports.com/es/reports/call-center-ai-market>

Alpha Cephei. (n.d.). Vosk offline speech recognition API.
<https://alphacephei.com/vosk/>

Google. (n.d.). Gemini API documentation. Google AI for Developers.
<https://ai.google.dev/gemini-api/docs>

Issabel. (n.d.). Issabel. <https://www.issabel.com/>

OHF-Voice. (n.d.). Piper1-gpl. GitHub. <https://github.com/OHF-Voice/piper1-gpl>

The Asterisk Project. (n.d.). Asterisk documentation. <https://docs.asterisk.org/>

Alpha Cephei. (n.d.). Vosk offline speech recognition API.
<https://alphacephei.com/vosk/>

Google. (n.d.). Gemini API documentation. Google AI for Developers.
<https://ai.google.dev/gemini-api/docs>

Issabel. (n.d.). Issabel. <https://www.issabel.com/>

OHF-Voice. (n.d.). Piper1-gpl. GitHub. <https://github.com/OHF-Voice/piper1-gpl>

The Asterisk Project. (n.d.). Asterisk documentation. <https://docs.asterisk.org/>

8. Anexos

Anexo 1: Código fuente de programación para transcripción de voz a texto

Archivo: `transcriber_service.py`

El código presentado es el encargado de la transcripción de audio a texto dentro de la arquitectura del asistente conversacional implementado. Su función principal consiste en recibir un archivo de audio en formato WAV mediante una solicitud HTTP, procesarlo con el motor de reconocimiento de voz Vosk y devolver como resultado una cadena de texto en formato JSON.

El módulo ha sido desarrollado en Python y se apoya en un conjunto de librerías orientadas al manejo de archivos, configuración del entorno, procesamiento de audio y despliegue web. En primer lugar, la instrucción `from __future__ import annotations` permite utilizar anotaciones de tipos de forma diferida, lo que mejora la compatibilidad y claridad del

código. La librería json se emplea para interpretar las respuestas generadas por Vosk, las cuales son devueltas en formato JSON. La librería os permite acceder a variables de entorno, mientras que tempfile facilita la creación de archivos temporales para almacenar los audios recibidos durante cada solicitud. Por su parte, unicodedata se utiliza para limpiar el texto transcrito, eliminando caracteres de control o símbolos no visibles que puedan afectar el resultado. La librería wave es necesaria para abrir y leer archivos de audio WAV, y Path, proveniente de pathlib, permite administrar rutas del sistema de archivos de manera más segura y portable.

```
from __future__ import annotations

import json
import os
import tempfile
import unicodedata
import wave
from pathlib import Path

from dotenv import load_dotenv
from flask import Flask, jsonify, request
from vosk import KaldiRecognizer, Model
from waitress import serve

BASE_DIR = Path(__file__).resolve().parents[1]
load_dotenv(BASE_DIR / ".env")

HOST = os.getenv("TRANSCRIBE_HOST", "0.0.0.0")
PORT = int(os.getenv("TRANSCRIBE_PORT", "7880"))
UPLOAD_DIR = Path(os.getenv("UPLOAD_DIR", "runtime/uploads"))
if not UPLOAD_DIR.is_absolute():
    UPLOAD_DIR = BASE_DIR / UPLOAD_DIR

VOSK_MODEL_PATH = os.getenv("VOSK_MODEL_PATH", "").strip()
if not VOSK_MODEL_PATH:
    raise RuntimeError("VOSK_MODEL_PATH is required")

MODEL = Model(VOSK_MODEL_PATH)
UPLOAD_DIR.mkdir(parents=True, exist_ok=True)

app = Flask(__name__)
```

```

def clean_text(text: str) -> str:
    return "".join(char for char in text if unicodedata.category(char)[0] !=
"C").strip()

def transcribe_audio(audio_path: Path) -> str:
    with wave.open(str(audio_path), "rb") as wav_file:
        recognizer = KaldiRecognizer(MODEL, wav_file.getframerate())
        parts: list[str] = []

        while True:
            data = wav_file.readframes(4000)
            if not data:
                break
            if recognizer.AcceptWaveform(data):
                result = json.loads(recognizer.Result())
                parts.append(result.get("text", ""))

            final_result = json.loads(recognizer.FinalResult())
            parts.append(final_result.get("text", ""))

        return clean_text(" ".join(part for part in parts if part))

@app.get("/health")
def health():
    return jsonify(
        {
            "status": "ok",
            "service": "transcriber",
            "port": PORT,
            "model_path": VOSK_MODEL_PATH,
        }
    )

@app.post("/transcribe")
def transcribe():
    if "audio" not in request.files:
        return jsonify({"error": "No audio file found"}), 400

    upload = request.files["audio"]
    suffix = Path(upload.filename or "input.wav").suffix or ".wav"

    with tempfile.NamedTemporaryFile(dir=UPLOAD_DIR, suffix=suffix,
delete=False) as temp_file:
        temp_path = Path(temp_file.name)
        upload.save(temp_path)

```

```

try:
    transcription = transcribe_audio(temp_path)
    return jsonify({"transcription": transcription})
finally:
    temp_path.unlink(missing_ok=True)

if __name__ == "__main__":
    serve(app, host=HOST, port=PORT)

```

Anexo 2: Código fuente del servicio de generación de respuestas con Gemini

Archivo: gemini_service.py

El código pertenece al módulo encargado de la generación de respuestas en lenguaje natural dentro de la arquitectura del asistente conversacional. El objetivo es recibir una consulta en texto, procesarla mediante el modelo generativo de Gemini y devolver una respuesta breve, clara y apta para ser reproducida posteriormente por el módulo de síntesis de voz. Es la etapa intermedia del sistema, ya que enlaza la transcripción obtenida desde el audio del usuario con la construcción de una respuesta dentro del flujo telefónico implementado.

De la misma manera este módulo ha sido desarrollado en Python y se apoya en varias librerías. La librería `datetime` se utiliza para obtener la fecha y hora actuales; se emplea `re` para trabajar con expresiones regulares y `ZoneInfo` posibilita definir con precisión la zona horaria en la que debe operar el sistema. Además, se emplea `load_dotenv` para cargar variables del archivo `.env`, `Flask`, `jsonify` y `request` para construir la API web, `genai` de Google para interactuar con la API de Gemini, y `waitress` para el servicio WSGI.

```

from __future__ import annotations

from datetime import datetime
import os
from pathlib import Path
import re
import unicodedata
from zoneinfo import ZoneInfo

from dotenv import load_dotenv
from flask import Flask, jsonify, request

```

```

from google import genai
from waitress import serve

BASE_DIR = Path(__file__).resolve().parents[1]
load_dotenv(BASE_DIR / ".env")

HOST = os.getenv("GEMINI_HOST", "0.0.0.0")
PORT = int(os.getenv("GEMINI_PORT", "7890"))
MODEL_NAME = os.getenv("GEMINI_MODEL", "gemini-3.5-flash")
API_KEY = os.getenv("GEMINI_API_KEY", "").strip()
TIMEZONE_NAME = os.getenv("APP_TIMEZONE", "America/Guayaquil").strip() or
"America/Guayaquil"
if not API_KEY:
    raise RuntimeError("GEMINI_API_KEY is required")

SYSTEM_PROMPT = (
    "Eres un asistente util y claro. "
    "Responde en una sola oracion, en espanol, con lenguaje sencillo y listo
para ser leído por voz. "
    "Si el usuario pregunta por fecha u hora, usa exclusivamente la fecha y
hora exactas que se te proporcionan en el contexto."
)

TIMEZONE = ZoneInfo(TIMEZONE_NAME)
client = genai.Client(api_key=API_KEY)
app = Flask(__name__)

SPANISH_WEEKDAYS = [
    "lunes",
    "martes",
    "miercoles",
    "jueves",
    "viernes",
    "sabado",
    "domingo",
]

SPANISH_MONTHS = [
    "",
    "enero",
    "febrero",
    "marzo",
    "abril",
    "mayo",
    "junio",
    "julio",
    "agosto",
    "septiembre",
    "octubre",

```

```

    "noviembre",
    "diciembre",
]

DATE_PATTERNS = [
    r"\bque fecha (es|es hoy|tenemos|estamos)\b",
    r"\bcual es la fecha\b",
    r"\bfecha (de hoy|actual)\b",
    r"\bque dia (es|es hoy|tenemos|estamos)\b",
    r"\bcual es el dia\b",
    r"\bdia de hoy\b",
    r"\bm es actual\b",
    r"\ba[nn]o actual\b",
    r"\ben que fecha estamos\b",
    r"\bhoy dia\b",
]

TIME_PATTERNS = [
    r"\bque hora es\b",
    r"\bcual es la hora\b",
    r"\bhora actual\b",
    r"\bhora de ahora\b",
    r"\bha que hora estamos\b",
]

def normalize_text(text: str) -> str:
    normalized = unicodedata.normalize("NFKD", text)
    ascii_text = normalized.encode("ascii", "ignore").decode("ascii")
    return re.sub(r"\s+", " ", ascii_text.lower()).strip()

def get_current_datetime() -> datetime:
    return datetime.now(TIMEZONE)

def build_date_phrase(now: datetime) -> str:
    weekday = SPANISH_WEEKDAYS[now.weekday()]
    month = SPANISH_MONTHS[now.month]
    return f"{weekday} {now.day} de {month} de {now.year}"

def build_time_phrase(now: datetime) -> str:
    return now.strftime("%H:%M")

def maybe_answer_datetime(user_text: str) -> str | None:
    normalized = normalize_text(user_text)
    asks_date = any(re.search(pattern, normalized) for pattern in
DATE_PATTERNS)

```

```

    asks_time = any(re.search(pattern, normalized) for pattern in
TIME_PATTERNS)

```

```

    if not asks_date and not asks_time:
        return None

```

```

    now = get_current_datetime()
    date_phrase = build_date_phrase(now)
    time_phrase = build_time_phrase(now)

```

```

    if asks_date and asks_time:
        return f"Hoy es {date_phrase} y son las {time_phrase}."
    if asks_time:
        return f"Son las {time_phrase}."
    return f"Hoy es {date_phrase}."

```

```

def ask_gemini(user_text: str) -> str:
    direct_answer = maybe_answer_datetime(user_text)
    if direct_answer is not None:
        return direct_answer

    now = get_current_datetime()
    prompt = (
        f"{SYSTEM_PROMPT}\n"
        f"Fecha actual exacta: {build_date_phrase(now)}.\n"
        f"Hora actual exacta: {build_time_phrase(now)}.\n"
        f"Zona horaria: {TIMEZONE_NAME}.\n\n"
        f"Usuario: {user_text}"
    )
    response = client.models.generate_content(
        model=MODEL_NAME,
        contents=prompt,
    )
    text = (response.text or "").strip()
    if not text:
        raise RuntimeError("Gemini returned an empty response")
    return text

```

```

@app.get("/health")
def health():
    return jsonify(
        {
            "status": "ok",
            "service": "gemini",
            "model": MODEL_NAME,
            "port": PORT,
            "timezone": TIMEZONE_NAME,

```

```

        "current_date": build_date_phrase(get_current_datetime()),
        "current_time": build_time_phrase(get_current_datetime()),
    }
)

@app.post("/gemini")
def gemini():
    data = request.get_json(silent=True) or {}
    user_text = str(data.get("texto", "")).strip()

    if not user_text:
        return jsonify({"error": "Missing 'texto' in request body"}), 400

    try:
        answer = ask_gemini(user_text)
    except Exception as error:
        return jsonify({"error": str(error)}), 500

    return jsonify(
        {
            "prompt": user_text,
            "response": answer,
        }
    )

if __name__ == "__main__":
    serve(app, host=HOST, port=PORT)

```

Anexo 3: Código fuente del servicio de síntesis de voz

Archivo: tts_service.py

Como una tercera etapa se realizó el módulo encargado de la síntesis de voz dentro de la arquitectura del asistente conversacional . Su función principal es recibir una respuesta textual generada por el sistema, convertirla en audio mediante el motor Piper y devolver un archivo WAV listo para ser reproducido en el entorno telefónico, ya que transforma la respuesta escrita en una señal de voz comprensible para el usuario.

Para su implementación se emplean librerías orientadas a configuración, procesamiento de audio y despliegue web. Flask permite construir la API del servicio;

waitress se utiliza para publicar la aplicación de forma estable; dotenv carga los parámetros del archivo .env; subprocess ejecuta el sintetizador Piper; soundfile y resample_poly de scipy.signal procesan el audio generado, permitiendo convertirlo a un formato compatible con telefonía. Este último punto es especialmente importante, ya que el sistema asegura que el audio final quede en mono y a 8 kHz, necesaria para su reproducción en Asterisk e Issabel.

```
from __future__ import annotations

from io import BytesIO
import math
import os
import re
import subprocess
import tempfile
from pathlib import Path

from dotenv import load_dotenv
from flask import Flask, jsonify, request, send_file
from scipy.signal import resample_poly
import soundfile as sf
from waitress import serve

BASE_DIR = Path(__file__).resolve().parents[1]
load_dotenv(BASE_DIR / ".env")

HOST = os.getenv("TTS_HOST", "0.0.0.0")
PORT = int(os.getenv("TTS_PORT", "7860"))
DEFAULT_VOICE = os.getenv("TTS_DEFAULT_VOICE", "audio_castillo1")

MODELS_DIR_RAW = os.getenv("TTS_MODELS_DIR", "").strip()
PIPER_EXE_RAW = os.getenv("PIPER_EXE", "").strip()
if not MODELS_DIR_RAW:
    raise RuntimeError("TTS_MODELS_DIR is required")
if not PIPER_EXE_RAW:
    raise RuntimeError("PIPER_EXE is required")

MODELS_DIR = Path(MODELS_DIR_RAW).expanduser()
PIPER_EXE = Path(PIPER_EXE_RAW).expanduser()
OUTPUT_DIR = Path(os.getenv("TTS_OUTPUT_DIR", "runtime/output"))

if not MODELS_DIR.is_absolute():
    MODELS_DIR = BASE_DIR / MODELS_DIR
if not PIPER_EXE.is_absolute():
```

```

    PIPER_EXE = BASE_DIR / PIPER_EXE
if not OUTPUT_DIR.is_absolute():
    OUTPUT_DIR = BASE_DIR / OUTPUT_DIR

OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

VOICE_MODELS = {
    "audio": "es_ES-davefx-medium.onnx",
    "audio_mujer": "es_ES-mls_10246-low.onnx",
    "audio_sefla": "es_MX-laura-high.onnx",
    "audio_juan": "es_LA-prueba_audio-high.onnx",
    "audio_castillo": "es_LA-prueba-high.onnx",
    "audio_castillo1": "es_prueba_audio-high.onnx",
    "audio_vanessa": "es_vanessa_audio-high.onnx",
    "audio_paula": "es_paula_audio-high.onnx",
}

TEXT_REPLACEMENTS = [
    ("(", ", "),
    (")", ", "),
    ("?", ", "),
    (";", ", "),
    (":", ", "),
]

app = Flask(__name__)

def normalize_text(text: str) -> str:
    clean = text.strip()
    for source, target in TEXT_REPLACEMENTS:
        clean = clean.replace(source, target)
    clean = re.sub(r"\s+", " ", clean)
    return clean[:10000]

def resample_to_8khz_mono(path: Path) -> None:
    audio, sample_rate = sf.read(path)

    if getattr(audio, "ndim", 1) > 1:
        audio = audio.mean(axis=1)

    if sample_rate != 8000:
        divisor = math.gcd(sample_rate, 8000)
        audio = resample_poly(audio, 8000 // divisor, sample_rate // divisor)

    sf.write(path, audio, 8000, subtype="PCM_16")

```

```

def resolve_voice_model(voice_alias: str) -> Path:
    model_name = VOICE_MODELS.get(voice_alias)
    if not model_name:
        raise ValueError(f"Unknown voice alias: {voice_alias}")

    model_path = MODELS_DIR / model_name
    if not model_path.exists():
        raise FileNotFoundError(f"Model not found: {model_path}")

    return model_path

def synthesize_to_wav(text: str, voice_alias: str) -> Path:
    if not PIPER_EXE.exists():
        raise FileNotFoundError(f"Piper executable not found: {PIPER_EXE}")

    model_path = resolve_voice_model(voice_alias)
    normalized_text = normalize_text(text)

    with tempfile.NamedTemporaryFile(dir=OUTPUT_DIR, suffix=".wav",
delete=False) as temp_file:
        output_path = Path(temp_file.name)

    try:
        subprocess.run(
            [str(PIPER_EXE), "-m", str(model_path), "-f", str(output_path)],
            input=normalized_text,
            text=True,
            capture_output=True,
            check=True,
        )
        resample_to_8khz_mono(output_path)
    except Exception:
        output_path.unlink(missing_ok=True)
        raise

    return output_path

@app.get("/health")
def health():
    return jsonify(
        {
            "status": "ok",
            "service": "tts",
            "default_voice": DEFAULT_VOICE,
            "voices": sorted(VOICE_MODELS.keys()),
            "port": PORT,
        }
    )

```

```

)

@app.post("/convert")
def convert():
    payload = request.get_json(silent=True) or {}
    text = str(payload.get("mensaje", "")).strip()
    voice = str(payload.get("modelo", DEFAULT_VOICE)).strip() or DEFAULT_VOICE

    if not text:
        return jsonify({"error": "Missing 'mensaje' in request body"}), 400

    try:
        output_file = synthesize_to_wav(text, voice)
        audio_bytes = output_file.read_bytes()
        download_name = output_file.name
    except Exception as error:
        return jsonify({"error": str(error)}), 500
    finally:
        if "output_file" in locals():
            output_file.unlink(missing_ok=True)

    return send_file(
        BytesIO(audio_bytes),
        mimetype="audio/wav",
        as_attachment=True,
        download_name=download_name,
    )

if __name__ == "__main__":
    serve(app, host=HOST, port=PORT)

```

Anexo 4: Archivo de configuración del entorno

Archivo: .env

Este archivo de configuración cumple la función dentro de la solución implementada, ya que centraliza los parámetros operativos del sistema, evita la codificación rígida de rutas y credenciales, y permite modificar el comportamiento de los servicios sin alterar directamente el código fuente.

```

GEMINI_API_KEY="aquí va la key"
GEMINI_MODEL=gemini-3.5-flash

```

```

GEMINI_HOST=0.0.0.0
GEMINI_PORT=7890
APP_TIMEZONE=America/Guayaquil

TRANSCRIBE_HOST=0.0.0.0
TRANSCRIBE_PORT=7880
UPLOAD_DIR=runtime/uploads
VOSK_MODEL_PATH="C:/Users/masqi/OneDrive/Escritorio/Trabajo de Titulacion
II/Capitulo 3/vosk-model-small-es-0.42/vosk-model-small-es-0.42"

TTS_HOST=0.0.0.0
TTS_PORT=7860
PIPER_EXE="C:/Users/masqi/OneDrive/Documentos/New
project/ivr_gemini_bootstrap/piper/piper.exe"
TTS_MODELS_DIR="C:/Users/masqi/OneDrive/Documentos/New
project/ivr_gemini_bootstrap/piper-models-ok"
TTS_DEFAULT_VOICE=audio
TTS_OUTPUT_DIR=runtime/output

```

Anexo 5: Código fuente del script AGI de integración telefónica

Archivo: agi.php

El código AGI permite integrar la central telefónica Issabel/Asterisk con los servicios externos de transcripción, generación de respuestas y síntesis de voz. Su función consiste en controlar el flujo completo de la llamada: contestar al usuario, grabar la consulta de voz, enviarla al servicio de transcripción, remitir el texto resultante al servicio Gemini, solicitar la respuesta sintetizada al módulo TTS y finalmente, reproducir el audio generado dentro de la misma llamada.

De la misma manera está desarrollado en PHP y utiliza la librería `phpagi.php`, la cual proporciona los métodos necesarios para interactuar con Asterisk mediante comandos AGI como Answer, Playback y Record. Asimismo, hace uso de `cURL` y del binario `/usr/bin/curl` para comunicarse con los servicios HTTP desplegados externamente, ubicados en los puertos 7880 para transcripción, 7890 para Gemini y 7860 para síntesis de voz. Gracias a ello, el script no realiza directamente el procesamiento inteligente, sino que coordina la comunicación entre la PBX y los microservicios de inteligencia artificial.

El funcionamiento general del programa se apoya en varias funciones auxiliares. La función `transcribirAudio` recibe el archivo grabado y lo envía al servicio de transcripción, devolviendo el texto detectado. La función `preguntarGemini` toma ese texto y solicita una respuesta al modelo generativo. La función `solicitarTTS` convierte la respuesta textual en audio WAV mediante el servicio Piper, mientras que `reproducirRespuestaTTS` guarda temporalmente dicho audio dentro del directorio de sonidos de Asterisk y lo reproduce al usuario, la función `esFraseSalida` permite detectar la expresión “muchas gracias”, la cual se utiliza como criterio de finalización voluntaria de la conversación.

```
#!/usr/bin/php -q
<?php
require('/var/lib/asterisk/agi-bin/phpagi.php');

$agi = new AGI();
$agi->answer();

function canalColgado($resultado) {
    return is_array($resultado)
        && isset($resultado['result'])
        && (string)$resultado['result'] === '-1';
}

function transcribirAudio($archivo, $agi) {
    $apiUrl = "http://192.168.0.102:7880/transcribe";
    $curlBin = "/usr/bin/curl";

    $agi->verbose("DEBUG: entrando a transcribirAudio con " . $archivo, 1);

    if (!file_exists($archivo)) {
        return "Error: no existe el archivo";
    }

    if (!file_exists($curlBin)) {
        return "Error: no existe /usr/bin/curl";
    }

    $cmd = $curlBin
        . " --connect-timeout 10 --max-time 40 -sS"
        . " -H " . escapeshellarg("Expect:")
        . " -X POST"
        . " -F " . escapeshellarg("audio=@" . $archivo . ";type=audio/wav")
```

```

        . " " . escapeshellarg($apiUrl)
        . " 2>&1";

$agi->verbose("DEBUG: ejecutando curl shell para transcripcion", 1);

$output = array();
$exitCode = 0;
exec($cmd, $output, $exitCode);

$response = trim(implode("\n", $output));

$agi->verbose("DEBUG: curl exit code: " . $exitCode, 1);
$agi->verbose("DEBUG: respuesta raw: " . $response, 1);

if ($exitCode !== 0) {
    return "Error en curl shell: " . $response;
}

$decoded = json_decode($response, true);

if (isset($decoded['transcripcion'])) {
    return trim($decoded['transcripcion']);
}

return "Error en la transcripcion";
}

function preguntarGemini($texto, $agi) {
    $apiUrl = "http://192.168.0.102:7890/gemini";

    if (!function_exists('curl_init')) {
        return "Error en Gemini: PHP cURL no esta disponible";
    }

    $payload = json_encode(array(
        'texto' => $texto
    ));

    $curl = curl_init();
    curl_setopt($curl, CURLOPT_URL, $apiUrl);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($curl, CURLOPT_POST, true);
    curl_setopt($curl, CURLOPT_POSTFIELDS, $payload);
    curl_setopt($curl, CURLOPT_HTTPHEADER, array(
        "Accept: application/json",
        "Content-Type: application/json",
        "User-Agent: PHP-cURL"
    ));
    curl_setopt($curl, CURLOPT_TIMEOUT, 60);

```

```

$response = curl_exec($curl);
$error = curl_error($curl);
curl_close($curl);

if ($error) {
    return "Error en Gemini: " . $error;
}

$debug->verbose("DEBUG GEMINI raw: " . $response, 1);

$decoded = json_decode($response, true);

if (isset($decoded['response'])) {
    return trim($decoded['response']);
}

return "Error en Gemini";
}

function solicitarTTS($texto, $agi) {
    $apiUrl = "http://192.168.0.102:7860/convert";

    if (!function_exists('curl_init')) {
        return array(false, "Error en TTS: PHP cURL no esta disponible");
    }

    $payload = json_encode(array(
        'mensaje' => $texto,
        'modelo' => 'audio'
    ));

    $curl = curl_init();
    curl_setopt($curl, CURLOPT_URL, $apiUrl);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($curl, CURLOPT_POST, true);
    curl_setopt($curl, CURLOPT_POSTFIELDS, $payload);
    curl_setopt($curl, CURLOPT_HTTPHEADER, array(
        "Accept: audio/wav,application/octet-stream",
        "Content-Type: application/json",
        "User-Agent: PHP-cURL"
    ));
    curl_setopt($curl, CURLOPT_TIMEOUT, 60);

    $response = curl_exec($curl);
    $error = curl_error($curl);
    $status = curl_getinfo($curl, CURLINFO_HTTP_CODE);
    $contentType = curl_getinfo($curl, CURLINFO_CONTENT_TYPE);
    curl_close($curl);

    if ($error) {

```

```

    return array(false, "Error en TTS: " . $err);
}

$agi->verbose("DEBUG TTS status: " . $status, 1);
$agi->verbose("DEBUG TTS content-type: " . $contentType, 1);

if ($status >= 400) {
    return array(false, "Error TTS HTTP " . $status);
}

if ($response === false || strlen($response) === 0) {
    return array(false, "Error TTS: audio vacio");
}

return array(true, $response);
}

function reproducirRespuestaTTS($wavBinary, $agi) {
    $baseName = "tts_respuesta_" . time();
    $wavPath = "/var/lib/asterisk/sounds/" . $baseName . ".wav";

    $bytes = file_put_contents($wavPath, $wavBinary);
    if ($bytes === false) {
        return "Error al guardar WAV";
    }

    clearstatcache();

    $agi->verbose("DEBUG TTS archivo guardado: " . $wavPath, 1);
    $agi->verbose("DEBUG TTS tamaño archivo: " . filesize($wavPath), 1);

    $agi->exec("Playback", $baseName);

    if (file_exists($wavPath)) {
        unlink($wavPath);
    }

    return "";
}

function esFraseSalida($texto) {
    $texto = strtolower(trim($texto));
    return strpos($texto, "muchas gracias") !== false;
}

$agi->verbose("DEBUG: iniciando modo conversacional", 1);

while (true) {
    $archivoBase = "/var/spool/asterisk/monitor/prueba_audio_" . time();
    $wav = $archivoBase . ".wav";

```

```

$agi->verbose("DEBUG: nueva vuelta de conversacion", 1);
$agi->verbose("DEBUG: archivo esperado " . $wav, 1);

$playBeep = $agi->exec("Playback", "beep");
if (canalColgado($playBeep)) {
    $agi->verbose("DEBUG: el usuario colgo antes de grabar", 1);
    break;
}

$recordResult = $agi->exec("Record", $wav . ",5,10");
if (canalColgado($recordResult)) {
    $agi->verbose("DEBUG: el usuario colgo durante la grabacion", 1);
    break;
}

usleep(500000);
clearstatcache();

$agi->verbose("DEBUG: existe archivo? " . (file_exists($wav) ? "si" : "no"), 1);

if (!file_exists($wav) || filesize($wav) < 1000) {
    $agi->verbose("DEBUG: no se obtuvo audio util, esperando otra consulta", 1);
    continue;
}

$agi->verbose("DEBUG: tamano archivo " . filesize($wav), 1);

$texto = transcribirAudio($wav, $agi);
$agi->verbose("Transcripcion detectada: " . $texto, 1);

if ($texto == "" || strpos($texto, "Error") === 0) {
    $agi->exec("Playback", "pbx-invalid");
    continue;
}

if (esFraseSalida($texto)) {
    $agi->exec("Playback", "goodbye");
    break;
}

$respuesta = preguntarGemini($texto, $agi);
$agi->verbose("Respuesta Gemini: " . $respuesta, 1);

if ($respuesta == "" || strpos($respuesta, "Error") === 0) {
    $agi->exec("Playback", "pbx-invalid");
    continue;
}

list($okTts, $ttsResultado) = solicitarTTS($respuesta, $agi);

```

```

if (!$okTts) {
    $agi->verbose($ttsResultado, 1);
    $agi->exec("Playback", "pbx-invalid");
    continue;
}

$errorPlayback = reproducirRespuestaTTS($ttsResultado, $agi);
if ($errorPlayback != "") {
    $agi->verbose($errorPlayback, 1);
    $agi->exec("Playback", "pbx-invalid");
    continue;
}

$agi->verbose("DEBUG: respuesta reproducida, esperando siguiente consulta", 1);
}

exit(0);
?>

```

Anexo 6: Pruebas realizadas a el sistema usando 10 preguntas sencillas

Como parte del proceso de validación del desempeño del sistema, se plantearon 10 preguntas sencillas, las cuales se detallan en la Tabla 7. A través de estas interacciones fue posible comprobar el correcto funcionamiento e integración de los módulos que conforman la arquitectura propuesta. En la primera prueba, correspondiente al saludo inicial, el sistema reconoció la entrada de voz del usuario, la transcribió correctamente y la envió al módulo de procesamiento. Posteriormente, la API de Gemini interpretó la consulta y generó una respuesta, la cual fue convertida automáticamente en voz mediante el módulo TTS, produciendo así el saludo y mensaje de bienvenida de forma satisfactoria.

Este mismo procedimiento se replicó en las nueve preguntas restantes, permitiendo verificar la continuidad del flujo conversacional entre el reconocimiento de voz, el procesamiento del lenguaje natural y la síntesis de audio. Además, se efectuó un máximo de 30 pruebas con el propósito de disponer de un conjunto de muestras suficiente para calcular tiempos de procesamiento y analizar el comportamiento general del sistema. Finalmente, se

estableció el comando “**Muchas gracias**” como instrucción de cierre de la sesión conversacional, aunque la llamada también podía finalizar manualmente por decisión del usuario.

Figura 15

Pregunta 1-saludo

```
hola_mundo.php: Transcripción detectada: hola
hola_mundo.php: DEBUG GEMINI raw: prompt:hoia,response:Hola,
hola_mundo.php:
hola_mundo.php: Respuesta Gemini: Hola, ¿en qué te puedo ayudar hoy?
hola_mundo.php: DEBUG TTS status: 200
hola_mundo.php: DEBUG TTS content-type: audio/wav
hola_mundo.php: DEBUG TTS archivo guardado: /var/lib/asterisk/sounds/tts_respuesta_1784219965.wav
hola_mundo.php: DEBUG TTS tamaño archivo: 32410
-- <SIP/100-00000004> Playing 'tts_respuesta_1784219965.slin' (escape_digits=) (sample_offset 0) (language 'en')
hola_mundo.php: DEBUG: archivo esperado /var/spool/asterisk/monitor/prueba_audio_1784219967.wav
-- AGI Script Executing Application: (Playback) Options: (beep)
-- <SIP/100-00000004> Playing 'beep.gsm' (language 'en')
-- AGI Script Executing Application: (Record) Options: (/var/spool/asterisk/monitor/prueba_audio_1784219967.wav,5,10)
-- <SIP/100-00000004> Playing 'beep.gsm' (language 'en')
[2026-07-16 11:39:33] NOTICE[2514]: chan_sip.c:15985 sip_reg_timeout: -- Registration for 'otavalo@10.0.60.16' timed out, trying again (Attempt 1)
hola_mundo.php: DEBUG: entrando a transcribirAudio con /var/spool/asterisk/monitor/prueba_audio_1784219967.wav
```

Figura 16

Pregunta 2

```
hola_mundo.php: Transcripción detectada: cómo estás
hola_mundo.php: DEBUG GEMINI raw: prompt:cu0013mo
hola_mundo.php:
hola_mundo.php: Respuesta Gemini: Hola, me encuentro muy bien y listo para ayudarte en lo que necesites hoy.
hola_mundo.php: DEBUG TTS status: 200
hola_mundo.php: DEBUG TTS content-type: audio/wav
hola_mundo.php: DEBUG TTS archivo guardado: /var/lib/asterisk/sounds/tts_respuesta_1784219990.wav
hola_mundo.php: DEBUG TTS tamaño archivo: 59902
-- <SIP/100-00000004> Playing 'tts_respuesta_1784219990.slin' (escape_digits=) (sample_offset 0) (language 'en')
[2026-07-16 11:39:53] NOTICE[2514]: chan_sip.c:15985 sip_reg_timeout: -- Registration for 'otavalo@10.0.60.16' timed out, trying again (Attempt 1)
hola_mundo.php: DEBUG: archivo esperado /var/spool/asterisk/monitor/prueba_audio_1784219996.wav
-- AGI Script Executing Application: (Playback) Options: (beep)
-- <SIP/100-00000004> Playing 'beep.gsm' (language 'en')
-- AGI Script Executing Application: (Record) Options: (/var/spool/asterisk/monitor/prueba_audio_1784219996.wav,5,10)
-- <SIP/100-00000004> Playing 'beep.gsm' (language 'en')
```

Figura 17

Pregunta 3

```
hola_mundo.php: Transcripción detectada: hora
hola_mundo.php: DEBUG GEMINI raw: prompt:hora,response:La
hola_mundo.php:
hola_mundo.php: Respuesta Gemini: La hora actual exacta es las dieciocho horas con cincuenta y siete minutos.
[2026-07-16 11:40:13] NOTICE[2514]: chan_sip.c:15985 sip_reg_timeout: -- Registration for 'otavalo@10.0.60.16' timed out, trying again (Attempt 1)
hola_mundo.php: DEBUG TTS status: 200
hola_mundo.php: DEBUG TTS content-type: audio/wav
hola_mundo.php: DEBUG TTS archivo guardado: /var/lib/asterisk/sounds/tts_respuesta_1784220017.wav
hola_mundo.php: DEBUG TTS tamaño archivo: 68076
```

Figura 18*Pregunta 4*

```

hola_mundo.php: Transcripcion detectada: día hoy
hola_mundo.php: DEBUG GEMINI raw: {prompt:du00eda
hola_mundo.php:
hola_mundo.php: Respuesta Gemini: Hoy es viernes 17 de julio de 2026.
hola_mundo.php: DEBUG TTS status: 200
hola_mundo.php: DEBUG TTS content-type: audio/wav
hola_mundo.php: DEBUG TTS archivo guardado: /var/lib/asterisk/sounds/tts_respuesta_1784220046.wav
hola_mundo.php: DEBUG TTS tamaño archivo: 54700

```

Figura 19*Pregunta 5*

```

hola_mundo.php:
hola_mundo.php: Transcripcion detectada: cuánto es dos más dos
hola_mundo.php: DEBUG GEMINI raw: {prompt:cuu00einto
hola_mundo.php:
hola_mundo.php: Respuesta Gemini: Dos más dos es igual a cuatro.
hola_mundo.php: DEBUG TTS status: 200
hola_mundo.php: DEBUG TTS content-type: audio/wav
hola_mundo.php: DEBUG TTS archivo guardado: /var/lib/asterisk/sounds/tts_respuesta_1784220071.wav
hola_mundo.php: DEBUG TTS tamaño archivo: 31852

```

Figura 20*Pregunta 6*

```

hola_mundo.php:
hola_mundo.php: Transcripcion detectada: de qué color el sol
hola_mundo.php: DEBUG GEMINI raw: {prompt:de
hola_mundo.php:
hola_mundo.php: Respuesta Gemini: Aunque desde la Tierra lo vemos de color amarillo, el sol es en realidad de color blanco.
hola_mundo.php: DEBUG TTS status: 200
hola_mundo.php: DEBUG TTS content-type: audio/wav
hola_mundo.php: DEBUG TTS archivo guardado: /var/lib/asterisk/sounds/tts_respuesta_1784220093.wav
hola_mundo.php: DEBUG TTS tamaño archivo: 80336

```

Figura 21*Pregunta 7*

```

hola_mundo.php: Transcripcion detectada: cuál es tu nombre
hola_mundo.php: DEBUG GEMINI raw: {prompt:cuu00e1l
hola_mundo.php:
hola_mundo.php: Respuesta Gemini: Hola, soy tu asistente virtual y estoy aquí para ayudarte con lo que necesites hoy.
[2026-07-16 11:41:53] NOTICE[2514]: chan_sip.c:15985 sip_reg_timeout: -- Registration for 'otavalo@10.0.60.16' timed out, trying again (Attempt
hola_mundo.php: DEBUG TTS status: 200
hola_mundo.php: DEBUG TTS content-type: audio/wav
hola_mundo.php: DEBUG TTS archivo guardado: /var/lib/asterisk/sounds/tts_respuesta_1784220115.wav

```

Figura 22*Pregunta 8*

```

hola_mundo.php:
hola_mundo.php: Transcripcion detectada: que es internet
hola_mundo.php: DEBUG GEMINI raw: {prompt:que
hola_mundo.php:
hola_mundo.php: Respuesta Gemini: El internet es una red mundial de computadoras conectadas entre sí que nos permite compartir información y com
hola_mundo.php: DEBUG TTS status: 200
hola_mundo.php: DEBUG TTS content-type: audio/wav
hola_mundo.php: DEBUG TTS archivo guardado: /var/lib/asterisk/sounds/tts_respuesta_1784220169.wav
hola_mundo.php: DEBUG TTS tamaño archivo: 149996

```

Figura 23*Pregunta 9*

```

hola_mundo.php: Transcripcion detectada: el fibra óptica
[2026-07-16 11:43:13] NOTICE[2514]: chan_sip.c:15985 sip_reg_timeout: -- Registration for 'otavalo@10.0.60.16' timed out, trying again (Attempt 1)
hola_mundo.php: DEBUG GEMINI raw: {prompt:el
hola_mundo.php: Respuesta Gemini: La fibra óptica es una tecnología que transmite datos a gran velocidad a través de hilos de vidrio muy delgado
hola_mundo.php: DEBUG TTS status: 200
hola_mundo.php: DEBUG TTS content-type: audio/wav

```

Figura 24*Pregunta 10*

```

hola_mundo.php: Transcripcion detectada: es una señal digital
hola_mundo.php: DEBUG GEMINI raw: {prompt:es
hola_mundo.php: Respuesta Gemini: Una señal digital es aquella que transmite información utilizando valores discretos y precisos, como los ceros
[2026-07-16 11:43:53] NOTICE[2514]: chan_sip.c:15985 sip_reg_timeout: -- Registration for 'otavalo@10.0.60.16' timed out, trying again (Attempt 1)
hola_mundo.php: DEBUG TTS status: 200
hola_mundo.php: DEBUG TTS content-type: audio/wav
hola_mundo.php: DEBUG TTS archivo guardado: /var/lib/asterisk/sounds/tts_respuesta_1784220234.wav
hola_mundo.php: DEBUG TTS tamano archivo: 146466

```

Figura 25*Palabra clave de cierre de cesión (Muchas gracias)*

```

hola_mundo.php:
hola_mundo.php: Transcripcion detectada: muchas gracias
-- AGI Script Executing Application: (Playback) Options: (goodbye)
-- <SIP/100-00000004> Playing 'goodbye.gsm' (language 'en')
-- <SIP/100-00000004>AGI Script hola_mundo.php completed, returning 4
== Spawn extension (from-internal, 800, 3) exited non-zero on 'SIP/100-00000004'
-- Executing [h@from-internal:1] Hangup("SIP/100-00000004", "") in new stack
== Spawn extension (from-internal, h, 1) exited non-zero on 'SIP/100-00000004'
issabel*CLI>
Disconnected from Asterisk server
Asterisk cleanly ending (0).
Executing last minute cleanups

```