



ESCUELA DE INGENIERÍA
INFORMÁTICA - GUAYANA

Universidad Católica Andrés Bello

Facultad de Ingeniería

Escuela de Ingeniería Informática

Sistema de Monitoreo Acústico, para Identificar Sonidos y Generar Alertas de Emergencia

Trabajo de Grado

presentado ante la

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

como parte de los requisitos para optar al título de

INGENIERO EN INFORMÁTICA

Realizado por

Naim, Carmelo

Tutor

Sotillo, César

Fecha

Lárez, Jesús

Marzo, 2025

Índice de Contenido

Resumen	IX
Introducción	1
Capítulo I. El Problema	3
Planteamiento del Problema	3
Objetivos	5
Objetivo General	5
Objetivos Específicos	5
Alcance	5
Limitaciones	6
Justificación	7
Capítulo II. Marco Teórico	8
Antecedentes de la Investigación	8
Real-time Audio Classification on an Edge Device [Clasificación de audio en tiempo real en un dispositivo perimetral]	8
Analysing RMS and peak values of vibration signals for condition monitoring of wind turbine gearboxes [Análisis de valores RMS y pico de señales de vibración para la monitorización del estado de las cajas de engranajes de aerogeneradores]	9
Metodología para la identificación de eventos sonoros anómalos	10
Edge AI for Real-Time Anomaly Detection in Smart Homes [Inteligencia artificial de borde para la detección de anomalías en tiempo real en hogares inteligentes]	10
Bayesian Optimization with Inequality Constraints [Optimización Bayesiana con Restricciones de Desigualdad]	11
Bases Teóricas	12
Dispositivos Edge	12
Raspberry pi	12
Python	13

Tensorflow	13
NestJs	14
Inteligencia artificial	14
Cadenas de Markov	15
Agente	17
Ciencia de datos	17
Técnicas predictivas para series temporales	18
LSTM (Long Short-Term Memory)	19
Autoencoders	20
Isolation Forest (Bosque de Aislamiento)	20
Yamnet	21
Redis Pub/Sub	22
Embedding computing (Computación Embebida)	22
Bases Legales	22
Capítulo III. Marco Metodológico	24
Tipo de Investigación	24
Metodología de Desarrollo Utilizada	24
Capítulo IV. Desarrollo y Resultados	28
Análisis inicial del sistema de monitoreo acústico para generar alertas en casos de emergencia.	28
Requerimientos Funcionales	30
Requerimientos no funcionales	30
Procedimiento Metodológico	30
Primera Iteracion	30
Segunda Iteracion	32
Tercera iteración	33
Cuarta Iteración	34
Quinta Iteración	36
Evaluacion de los Modelos	41
Evaluacion del LSTM Autoencoder	41

Evaluacion del Isolation Forest	45
Sexta Iteración	49
Evaluacion del LSTM Autoencoder V2	50
Diseño final de la arquitectura del sistema	53
Selección de hardware y dispositivos Edge	56
Capítulo V. Conclusiones y Recomendaciones	59
Conclusiones	59
Recomendaciones	60
Referencias	62
Apéndices	67
Apéndice A. Representación cíclica del tiempo mediante funciones trigonométricas	68
Apéndice B. Algorimos de los distintos agentes	70
Apéndice C. Verificación del sistema de monitoreo acústico	73
Apéndice D. Cuadro comparativo de microcontroladores	75
Apéndice E. Manual de Usuario	76
Apéndice F. Manual de Sistema	80
Anexos	91
Anexo A. Dispositivo Raspberry Pi 4 Model B	92
Anexo B. Construcción del Isolation Forest (Bosque de Aislamiento)	93
Anexo C. Numero de datos de entrenamiento por categoría de sonido	94
Anexo D. Categorías de sonido del modelo YAMNet	95

Índice de Tablas

Tabla 1. Cuadro Comparativo de metodologías	25
Tabla 2. Matriz de Confusión del LSTM Autoencoder	43
Tabla 3. Matriz de Confusión del Isolation Forest	47
Tabla 4. Matriz de Confusión del LSTM Autoencoder V2	52
Tabla 5. Requerimientos del sistema acústico	74
Tabla 6. Cuadro comparativo de microcontroladores	75
Tabla 21. Lista de Índices y Categorías de Sonido	95

Índice de Figuras

Figura 1. Modelo de desarrollo en espiral propuesto por Boehm (1988)	27
Figura 2. Diagrama de arquitectura de sistema con ESP32 y detección de intensidad de sonido	31
Figura 3. Diagrama de arquitectura de sistema con ESP32, micrófono de señales de audio y prophet	32
Figura 4. Diagrama de arquitectura de sistema con Raspberry Pi, Yamnet y Cadenas de Markov	34
Figura 5. Diagrama de arquitectura de sistema con Raspberry Pi Autonomo, Yamnet y Modelos de IA para detección de anomalías	36
Figura 6. Visualización del dataset de secuencias para el entrenamiento de los Modelos .	39
Figura 7. Curva ROC y AUC del LSTM Autoencoder	42
Figura 8. Gráficas de MSE por secuencia para el LSTM Autoencoder.	42
Figura 9. Gráficas de puntuación de anomalía por evento para el Isolation Forest.	46
Figura 10. Curva ROC y AUC del Isolation Forest.	48
Figura 11. Parametros originales.	49
Figura 12. Parametros nuevos.	50
Figura 13. Curva ROC y AUC del LSTM Autoencoder V2.	51
Figura 14. Gráficas de MSE por secuencia para el LSTM Autoencoder V2.	51
Figura 15. Arquitectura del sistema de monitoreo acústico para generar alertas en casos de emergencia	54
Figura 16. Ejemplo de notificaciones.	58
Figura 17. Visualización de las funciones seno y coseno	68
Figura 18. Visualización discreta del comportamiento cíclico del tiempo	68
Figura 19. Curva paramétrica continua del tiempo sobre el círculo unitario	69
Figura 20. Algoritmo del agente de detección de anomalías basado en Isolation Forest .	70
Figura 21. Algoritmo del agente de predicción basado en LSTM	71
Figura 22. Algoritmo del agente de detección de silencio	72

Figura 23. Arquitectura del Raspberry Pi 4 Model B	92
Figura 24. Algoritmo de construcción del Isolation Forest	93
Figura 25. Número de datos de entrenamiento por categoría de sonido	94

Índice de Fórmulas

Fórmula 1. Accuracy para LSTM Autoencoder	44
Fórmula 2. Tasa de Error para LSTM Autoencoder	44
Fórmula 3. Recall para LSTM Autoencoder	44
Fórmula 4. Precision para LSTM Autoencoder	44
Fórmula 5. Specificity para LSTM Autoencoder	44
Fórmula 6. F1 Score para LSTM Autoencoder	45
Fórmula 7. Accuracy para Isolation Forest	47
Fórmula 8. Tasa de Error para Isolation Forest	47
Fórmula 9. Recall para Isolation Forest	47
Fórmula 10. Precision para Isolation Forest	47
Fórmula 11. Specificity para Isolation Forest	47
Fórmula 12. F1 Score para Isolation Forest	48
Fórmula 13. Accuracy para LSTM Autoencoder V2	52
Fórmula 14. Tasa de Error para LSTM Autoencoder V2	52
Fórmula 15. Recall para LSTM Autoencoder V2	52
Fórmula 16. Precision para LSTM Autoencoder V2	52
Fórmula 17. Specificity para LSTM Autoencoder V2	53

Universidad Católica Andrés Bello

Facultad de Ingeniería

Escuela de Ingeniería Informática

Sistema de Monitoreo Acústico, para Identificar Sonidos y Generar Alertas de Emergencia

Autor: Naim, Carmelo

Sotillo, César

Tutor Académico: Lárez, Jesús

Fecha: Marzo, 2025

Resumen

En el presente trabajo se aborda el desarrollo de un sistema de monitoreo acústico basado en Inteligencia Artificial que emite alertas en tiempo real al identificar eventos anómalos en sonidos ambientales que pueden indicar que una persona se encuentra en situación de vulnerabilidad. Para el desarrollo del sistema se utilizaron dispositivos edge Raspberry Pi 4 y un servidor. Se diseñó una arquitectura de IA que empleó los modelos preentrenados YAMNet para clasificación de sonidos ambientales y Vosk para detección de palabras clave. Posteriormente, se aplicaron modelos avanzados de aprendizaje automático (Isolation Forest, LSTM) para el perfilado del comportamiento acústico y la detección de patrones anómalos a partir de dichas clasificaciones. Para abordar los diferentes desafíos técnicos y la incertidumbre inherente al proyecto, se empleó una metodología basada en el Modelo Espiral, la cual permitió un desarrollo iterativo a través de las fases de Planificación, Análisis y mitigación de riesgos, Desarrollo incremental y evaluación, y Planificación de la siguiente iteración. La solución obtenida se sustenta en el rendimiento sobresaliente del modelo LSTM, que demostró una Exactitud del 99.98, Precision del 90.48, Recall de 1 en la detección de anomalías, genera notificaciones automáticas a contactos de emergencia ante eventos críticos.

Palabras clave— Clasificación de secuencias acústicas anómalas, Emisión de alertas en tiempo real, Inteligencia Artificial, Monitoreo de sonidos ambientales, Sistemas Embebidos, Dispositivos Edge

Introducción

Actualmente, el desarrollo de sistemas de alerta temprana se ha convertido en una prioridad social y tecnológica debido a la detección tardía de accidentes o complicaciones de salud en individuos vulnerables que viven o pasan largos períodos de tiempo solos. Esta problemática se agudiza por el envejecimiento demográfico, la CEPAL proyecta que la población mayor de 60 años en América Latina se triplicará entre 2000 y 2050, y por la estructura familiar moderna, marcada por el aumento de hogares unipersonales. Esta confluencia de factores expone a las personas a múltiples riesgos, ya que la falta de asistencia inmediata en situaciones críticas puede transformar un incidente inicialmente manejable en una amenaza grave para la vida, incrementando las tasas de mortalidad y discapacidad asociadas.

Si bien existen soluciones tecnológicas, como dispositivos de detección de caídas integrados en wearables (e.g., Apple Watch) o dispositivos de alerta dedicados (collares o botones de emergencia), estas presentan limitaciones en cuanto a accesibilidad y cobertura. Su enfoque se centra en eventos puntuales y no proveen un monitoreo continuo y adaptable a diversas emergencias, siendo particularmente ineficaces si la persona se encuentra inconsciente o es sometida a coacción. Adicionalmente, su dependencia de la activación manual constituye una barrera tecnológica significativa para la población de edad avanzada o con condiciones limitantes.

Este trabajo aborda la necesidad de mecanismos efectivos de alerta temprana mediante el desarrollo de un sistema distribuido que utiliza Inteligencia Artificial para la detección de anomalías acústicas. El sistema está diseñado para servir a cualquier individuo en condición de vulnerabilidad. Su arquitectura funcional emplea una Raspberry Pi operando como sistema embebido para la captura y procesamiento inicial en el borde (Edge Computing), complementada por un servidor central que gestiona las alertas y comunicaciones externas, manteniendo el dispositivo de procesamiento de audio completamente aislado de internet. Se diseña una arquitectura de software que utiliza modelos preentrenados (YAMNet para detección de sonidos ambientales y Vosk para clasificación de palabras clave). Posteriormente, se aplican modelos avanzados de aprendizaje automático (Isolation Forest y LSTM) para el descubrimiento y clasificación de secuencias de patrones anómalos a partir de dichas detecciones, lo que permite al sistema caracterizar el comportamiento acústico normal del entorno e identificar desviaciones significativas que podrían indicar situaciones de emergencia.

Para la implementación, se emplea una metodología basada en el Modelo de Desarrollo en Espiral, un enfoque iterativo adecuado para proyectos con alta incertidumbre técnica. Esta metodología permitió un desarrollo efectivo mediante la iteración de las fases de Planificación, Análisis y mitigación de riesgos, Desarrollo incremental y evaluación, y Planificación de la siguiente iteración. La solución obtenida no solo cumple con los requerimientos funcionales establecidos, sino que garantiza la privacidad de los usuarios al realizar el procesamiento primario de los datos de audio de manera local. La importancia de este proyecto radica en su potencial para beneficiar a familias, cuidadores e instituciones, facilitando una intervención más oportuna ante diversas emergencias, incluyendo casos de violencia doméstica o el monitoreo de personas con depresión, contribuyendo así a una mejor calidad de vida.

El presente trabajo está estructurado en cinco capítulos: el Capítulo I describe el Planteamiento del Problema, donde se expone la problemática, los objetivos, justificación, alcance y limitaciones del proyecto; el Capítulo II expone el Marco Teórico, que recopila los antecedentes y bases teóricas que sustentan el trabajo; el Capítulo III presenta el Marco Metodológico, describiendo el tipo de investigación, técnicas e instrumentos de recolección de datos, metodología de desarrollo y procedimiento metodológico; el Capítulo IV detalla el Desarrollo y Resultados, exponiendo cómo el procedimiento metodológico dio respuesta a cada objetivo planteado; y el Capítulo V contiene las Conclusiones y Recomendaciones sobre el trabajo realizado. Finalmente, se listan las referencias bibliográficas, seguidas de los anexos y apéndices.

Capítulo I. El Problema

Planteamiento del Problema

En la actualidad, existe un número significativo de casos de personas que sufrieron algún accidente o complicación de una condición de salud, producto de la detección tardía de estos eventos, una tendencia que ha ido en aumento debido al envejecimiento de la población, que de acuerdo a los expuesto por la CEPAL (2005), dice que: “el envejecimiento demográfico en América Latina se triplicará entre el 2000 y 2050, con un aumento significativo en la proporción de personas mayores de 60 años” y a la estructura familiar moderna, que según Peña (2016), “El aumento de los hogares unipersonales se ha convertido en un fenómeno extendido dentro del mundo occidental desarrollado”. Esta situación, expone a las personas a múltiples riesgos, especialmente cuando se enfrentan a emergencias domésticas, problemas de salud o accidentes que comprometen su bienestar físico y emocional. La falta de asistencia inmediata en situaciones críticas puede transformar un incidente inicialmente manejable en una amenaza grave para la vida.

Algunos casos ilustran estos riesgos. Un ejemplo es el de un hombre octogenario en Zaragoza, España, cuyo fallecimiento en su domicilio pasó desapercibido durante una semana. No fue hasta que una vecina notó su ausencia que se alertó a las autoridades. Aunque la causa de su muerte fue natural, este caso pone en evidencia la vulnerabilidad de las personas que viven solas y la falta de un mecanismo que pueda detectar la inactividad prolongada o cambios abruptos en la rutina diaria (Ralla, 2024). En una situación como esta, el problema no es la atención médica, sino la ausencia de una alerta automática que notifique una anomalía grave (la total falta de actividad) a familiares o servicios de emergencia. De forma similar, en un caso no fatal como una caída, la falta de esa misma alerta inmediata sí puede derivar en graves complicaciones físicas, transformando un accidente manejable en una crisis de salud. Estos incidentes no son excepcionales, sino que representan una problemática real, especialmente en el contexto de los hogares unipersonales.

El envejecimiento de la población es un fenómeno natural que está incrementando la cantidad de personas mayores que viven en solitario. De acuerdo a lo comentado por la Naciones Unidas (2022): “se estima que para el 2050, el 16 % de la población mundial estará compuesta por personas mayores de 65 años, muchas de las cuales optarán o se verán forzadas a vivir sin compañía”. Esta tendencia plantea un desafío crítico en cuanto a su seguridad y bienestar. La falta de asistencia oportuna en emergencias no solo

representa una vulnerabilidad significativa para la vida de estas personas ante eventos que pongan en riesgo su vida o integridad física, sino que, a largo plazo, puede contribuir a un deterioro progresivo de su estado de salud y autonomía. Además, situaciones como caídas, ataques cardíacos o accidentes domésticos, que podrían ser atendidos a tiempo, se convierten en amenazas letales debido a la ausencia de un sistema de detección temprana.

Aunque en los últimos años se han desarrollado algunas soluciones tecnológicas para abordar esta problemática, como el dispositivo Zoe Fall, que detecta caídas a través de ondas WiFi, o el Apple Watch, que incluye una función de detección de caídas, estas tecnologías no son accesibles para todos ni cubren una gama suficientemente amplia de emergencias. De acuerdo a lo comentado por Carmona (2024): “estas herramientas están mayormente enfocadas en eventos puntuales, como caídas, pero no proporcionan un monitoreo continuo y adaptable a otras situaciones que podrían requerir detección temprana”. Adicionalmente, la mayoría de estos dispositivos dependen de la capacidad del usuario para activarlos o configurarlos adecuadamente, lo cual representa una barrera tecnológica en personas que, por su edad o condición, pueden no estar familiarizadas con el uso de la tecnología.

La falta de mecanismos efectivos para detectar emergencias a tiempo no solo prolonga el sufrimiento de las personas en situaciones críticas, sino que también incrementará las tasas de mortalidad y discapacidad asociadas a estos eventos. La inacción en este sentido implicaría ignorar las necesidades de una población vulnerable, cuya seguridad y calidad de vida pueden verse directamente influenciadas por la imposibilidad de notificar estos eventos.

En este contexto, se evidencia la necesidad de desarrollar una solución tecnológica que supere las limitaciones de los dispositivos actuales, ofreciendo un monitoreo más integral, no invasivo y accesible. Por lo tanto, este trabajo se centra en investigar y proponer un sistema de monitoreo inteligente basado en el análisis del entorno acústico. La pregunta central que guía esta investigación es: ¿Es posible, utilizando inteligencia artificial en dispositivos de bajo costo, crear un sistema que caracterice el comportamiento sonoro habitual de un hogar y genere alertas fiables ante patrones anómalos, respetando al mismo tiempo la privacidad del usuario?

Este sistema, ofrecerá un monitoreo continuo, creando un perfil acústico de cada ambiente del hogar que permitirá diferenciar entre sonidos típicos y atípicos en cada entorno.

Al clasificar un sonido que se salga de los patrones típicos, el sistema enviará una consulta al individuo para confirmar si está en buen estado; en caso de no recibir respuesta o de que se confirme una situación de emergencia, enviará una alerta inmediata a los contactos de emergencia predefinidos.

La solución permite caracterizar cada entorno del hogar, ya sea un hogar con una persona mayor, una persona con discapacidad, un hogar con muchos niños, o un entorno con vecinos que escuchan música alta con frecuencia. Este perfil facilita la detección de situaciones anómalas que puedan representar un riesgo, especialmente en viviendas donde viven personas solas, quienes pueden enfrentar desafíos por la falta de compañía o asistencia. Así, el sistema está diseñado no solo para adultos mayores o personas con discapacidad, sino para cualquier persona en situación de vulnerabilidad.

La solución propuesta tiene el potencial de reducir los riesgos asociados a la detección tardía de estos eventos, proporcionando a las personas un mayor nivel de seguridad sin comprometer su privacidad.

Objetivos

Objetivo General.

Desarrollar un sistema para generar alertas en casos de emergencia basado en el monitoreo acústico.

Objetivos Específicos.

1. Analizar los conceptos asociados a la analítica de sonidos ambientales identificando los conceptos necesarios para diseñar el sistema.
2. Diseñar un sistema para generar alertas en casos de emergencia basado en el monitoreo acústico en función del análisis realizado.
3. Implementar el sistema para generar alertas en casos de emergencia basado en el monitoreo acústico según el diseño realizado.
4. Validar el sistema para generar alertas en casos de emergencia basado en el monitoreo acústico con respecto al análisis realizado.
5. Elaborar la documentación técnica y guías del usuario y sistema.

Alcance

El proyecto tiene como objetivo desarrollar un sistema de monitoreo acústico que profile el comportamiento de los habitantes de una vivienda, a través del reconocimiento y clasificación de los

sonidos ambientales en las diferentes estancias de la misma. El sistema generará alarmas en tiempo real que alertarán a contactos de emergencia predefinidos. Para ello se llevará a cabo un análisis de los conceptos asociados a la analítica de sonidos donde se identificarán los conceptos necesarios para diseñar el sistema.

Con base en los resultados del análisis, se procede a diseñar el sistema de monitoreo acústico capaz de reconocer sonidos. El diseño comprende la definición de la arquitectura del sistema, contemplando los componentes esenciales, como la red de micrófonos para la captura de sonidos y los mecanismos de alerta, detección de anomalías y de reconocimiento acústico. El diseño considera la importancia de la privacidad de los usuarios. Según los resultados del análisis realizado para el desarrollo del sistema de monitoreo acústico, se evalúa si es necesario crear un dataset desde cero, obtener uno de internet y modificarlo según sea necesario, o bien emplear un modelo ya preentrenado.

Una vez completado el diseño, se procede a la implementación del sistema de monitoreo acústico. Esta fase incluye la configuración e instalación de los componentes físicos, como los micrófonos distribuidos en las estancias de la vivienda, asegurando una cobertura adecuada para la captura de sonidos relevantes. Los algoritmos de procesamiento de señales y los modelos de inteligencia artificial, definidos en la fase de diseño, fueron desarrollados y adaptados para realizar el reconocimiento y clasificación de los sonidos.

Se desarrollaron los mecanismos de alerta y notificación, que emiten avisos a los contactos de emergencia predefinidos.

Luego se realiza la validación del sistema, se lleva a cabo pruebas funcionales en ambientes controlados y no controlados para comparar los resultados obtenidos con los objetivos planteados y realizar ajustes en caso de ser necesario.

Finalmente se elabora la documentación del sistema, manuales de usuario, manuales de sistema, descripción de los componentes, diagramas y cualquier otro documento necesario.

Limitaciones

En la implementación del sistema, ciertas limitaciones que influyen en el desempeño de su desarrollo. A continuación, se describen algunas de las restricciones que se presentan:

Limitaciones tecnológicas: La precisión del sistema se ve afectada por la calidad de los sensores acústicos y la capacidad de procesamiento de los microcontroladores empleados.

Limitaciones de datos: La recolección de insuficientes muestras de sonidos para entrenar el modelo

limita su aprendizaje, lo que compromete el correcto funcionamiento del sistema en escenarios no previstos.

Limitaciones operativas: La inestabilidad del suministro eléctrico, especialmente en contextos donde se realizaron las pruebas, interrumpe la recolección continua de datos y afecta la disponibilidad del sistema para capturar eventos clave en su contexto temporal.

Limitaciones de hardware: El reentrenamiento de modelos complejos, como los basados en Transformers o LSTM, requiere recursos de cómputo significativos que superan la capacidad de dispositivos de bajo consumo como Raspberry Pi. Esto restringe la posibilidad de actualizar o adaptar los modelos directamente en el dispositivo, obligando a depender de un computador externo para estas tareas.

Justificación

En la actualidad, la atención a personas en condiciones de vulnerabilidad justifica el desarrollo de soluciones tecnológicas que permitan una respuesta rápida ante situaciones críticas, sin comprometer la privacidad ni depender de una supervisión constante.

Desde una perspectiva social, el sistema puede beneficiar a personas que, por condiciones físicas o emocionales, se encuentran en mayor riesgo de enfrentar emergencias sin posibilidad de pedir ayuda inmediata, como en casos de caídas, crisis de salud o situaciones de violencia doméstica. También contribuye a la labor de cuidadores y servicios de emergencia al ofrecer alertas tempranas.

En el ámbito institucional, el proyecto plantea una solución aplicable en hogares de cuidado, hospitales o comunidades, ya que su arquitectura de bajo costo y su implementación con componentes disponibles permiten su adaptación a contextos con recursos limitados. Esta característica puede ser relevante en la formulación de futuras regulaciones públicas que busquen integrar herramientas automatizadas en estrategias de atención ciudadana.

El sistema se vincula con los Objetivos de Desarrollo Sostenible (ODS), particularmente el ODS 3: Salud y bienestar, al aportar un mecanismo de alerta temprana en situaciones críticas, y el ODS 10: Reducción de desigualdades, al ofrecer una solución técnica accesible que no depende de infraestructura compleja. Su diseño se considera adaptable por ser independiente del ambiente donde se implemente, lo que lo hace funcional en diferentes contextos sociales, promoviendo mayor equidad en el acceso a herramientas de apoyo.

Capítulo II. Marco Teórico

Este capítulo presenta las bases teóricas y las investigaciones previas que sustentan el desarrollo del Sistema de Monitoreo Acústico para Identificar Sonidos y Generar Alertas de Emergencia. A través de la revisión de fuentes documentales, se exponen los conceptos y enfoques principales relacionados con el procesamiento de señales acústicas, la clasificación de sonidos mediante técnicas de inteligencia artificial y los sistemas de alertas automatizadas.

El marco teórico no solo proporciona un contexto académico para el proyecto, sino que también justifica las decisiones técnicas y metodológicas adoptadas en el diseño del sistema. En las siguientes secciones se detallan los antecedentes investigativos, los modelos de IA aplicados a la clasificación de sonidos y los fundamentos teóricos que permiten la generación de alertas en situaciones de emergencia.

En las siguientes secciones se detallan los antecedentes y los conceptos clave que sustentan este trabajo.

Antecedentes de la Investigación

Real-time Audio Classification on an Edge Device [Clasificación de audio en tiempo real en un dispositivo perimetral].

Malmberg y Radszuweit (2021) Abordan la implementación de modelos de aprendizaje automático en dispositivos edge para la clasificación de audio en tiempo real. Se centra en el uso del modelo YAMNet, que fue reentrenado para detectar eventos acústicos como disparos, rotura de cristales, animales y habla humana. Este modelo puede desplegarse en dispositivos edge en su versión completa con TensorFlow como en versiones compactas con TensorFlow Lite, con el objetivo de comparar la precisión, el tiempo de inferencia y el uso de memoria de cada variante.

Con el propósito de evaluar el desempeño, trabajaron en una serie de experimentos en los que compararon la precisión del modelo en ambas versiones de TensorFlow. Se encontraron con que, aunque existía una pérdida de precisión en la versión lite los resultados eran comparables a los de la versión completa. Esto implica que TensorFlow Lite es una opción viable y crea la posibilidad de trabajar con

dispositivos de bajo consumo y recursos limitados como ESP32

Este enfoque no solo demuestra la factibilidad de implementar modelos de clasificación de audio en tiempo real en entornos con recursos restringidos, sino que también abre la puerta a futuras investigaciones que busquen mejorar estos modelos. La capacidad de desplegar soluciones de inteligencia artificial en dispositivos edge amplía las posibilidades de aplicaciones en áreas como la seguridad, la vigilancia y el monitoreo ambiental, permitiendo respuestas más rápidas y reduciendo la dependencia de infraestructuras centralizadas.

Analysing RMS and peak values of vibration signals for condition monitoring of wind turbine gearboxes [Análisis de valores RMS y pico de señales de vibración para la monitorización del estado de las cajas de engranajes de aerogeneradores].

Igba, Alemzadeh, Durugbo, y Eiriksson (2016) abordan el monitoreo de condición en aerogeneradores con el objetivo de detectar fallos en las cajas de engranajes mediante el análisis de valores RMS y picos de vibraciones. La investigación propone tres modelos: correlación de señales, vibración extrema e intensidad RMS, validados con datos en dominio del tiempo. A través del uso de la teoría de valores extremos, se identificaron indicadores que permiten detectar fallos en etapas tempranas, lo que facilita la planificación del mantenimiento y minimiza el tiempo de inactividad de los aerogeneradores. Los resultados demostraron que el monitoreo de vibraciones proporciona información clave sobre el estado de los componentes y que la precisión de cada técnica depende de la física de la falla, sugiriendo un enfoque integral que combine distintas estrategias para una evaluación más robusta de la salud de los aerogeneradores.

El uso del valor RMS en este estudio es fundamental para la monitorización de la condición de las cajas de engranajes en aerogeneradores, ya que permite evaluar el nivel global de vibración y detectar fallos progresivos, como el desgaste de rodamientos y grietas en ejes. Igba y cols. (2016) destacan que el RMS es una métrica confiable para identificar tendencias anómalas en las vibraciones, lo que facilita la detección temprana de fallos antes de que se conviertan en problemas críticos. A pesar de ciertas limitaciones, como su menor sensibilidad a fallos incipientes en los dientes de los engranajes, el análisis de RMS sigue siendo un pilar clave en la estrategia de mantenimiento basado en condición (CBM), al proporcionar información valiosa sobre la evolución del estado de los componentes mecánicos.

Metodología para la identificación de eventos sonoros anómalos.

Torija, Ruiz, y Ramos-Diao (2018) presentan una metodología para la detección de eventos sonoros anómalos en entornos urbanos. Su trabajo se centra en el análisis de sucesos acústicos que generan incrementos bruscos de energía sonora en el paisaje sonoro urbano, lo que puede provocar molestias significativas en la población expuesta. Según los autores, los eventos sonoros anómalos, como bocinas, gritos, explosiones o disparos, generan una percepción intensificada del ruido ambiental debido a su impacto en la focalización de la atención. Para abordar este problema, desarrollaron un modelo basado en la evaluación del tiempo de estabilización de la medición del ruido ambiental y el análisis del factor cresta, que permite estimar el incremento de energía sonora generado por estos eventos.

El estudio incluyó mediciones en 35 localizaciones de la ciudad de Granada, utilizando un sonómetro Brüel Kjaer tipo 1. La metodología aplicada permitió definir un evento sonoro anómalo como aquel que provoca un incremento del nivel de energía sonora de al menos un 25 % respecto al nivel de fondo caracterizado por el descriptor LA90.

Los resultados indicaron que el número de eventos sonoros anómalos presentes en una ubicación está altamente correlacionado con la variabilidad de la energía sonora en la zona. Además, se evidenció que el factor cresta es un parámetro clave para estimar la magnitud del impacto acústico.

Edge AI for Real-Time Anomaly Detection in Smart Homes [Inteligencia artificial de borde para la detección de anomalías en tiempo real en hogares inteligentes].

Reis y Serôdio (2025) proponen un marco de trabajo para la detección de anomalías en tiempo real en dispositivos de borde (Edge AI), con el objetivo de superar las limitaciones de latencia y privacidad de los sistemas centralizados en la nube. Su enfoque se basa en una arquitectura híbrida que combina dos modelos de aprendizaje automático, Isolation Forest (IF), un modelo ligero para la detección rápida de eventos puntuales, y un Autoencoder de Memoria a Corto y Largo Plazo (LSTM-AE), un modelo de aprendizaje profundo para identificar patrones anómalos en secuencias de datos.

El sistema fue evaluado en plataformas de borde como Raspberry Pi y NVIDIA Jetson Nano para validar su desempeño en entornos con recursos limitados. Con el propósito de evaluar el rendimiento, los autores compararon la precisión de ambos modelos. Los resultados demostraron que el modelo LSTM alcanzó una mayor precisión de detección (hasta un 93.6 %), pero con un costo computacional y energético

más elevado. Un hallazgo clave fue la aplicación de técnicas de cuantización sobre el modelo LSTM, lo que permitió una reducción del 76 % en el tiempo de inferencia y del 35 % en el consumo de energía. Esto confirma que es viable trabajar con versiones ligeras de modelos complejos en dispositivos edge sin sacrificar significativamente la precisión.

Este enfoque no solo demuestra que es posible implementar sistemas de monitoreo de anomalías en tiempo real que sean rápidos, privados y de bajo consumo energético, sino que también subraya la ventaja de una arquitectura híbrida que equilibra velocidad y precisión. La capacidad de desplegar estas soluciones directamente en el hardware del hogar amplía sus aplicaciones a otros dominios como edificios inteligentes e IoT industrial, reduciendo la dependencia de la infraestructura en la nube y permitiendo respuestas más autónomas ante eventos críticos.

Bayesian Optimization with Inequality Constraints [Optimización Bayesiana con Restricciones de Desigualdad].

La Optimización Bayesiana (OB) es un marco de optimización secuencial diseñado para encontrar el mínimo o máximo de funciones que son costosas de evaluar, utilizando un número reducido de iteraciones. A diferencia de métodos tradicionales como la búsqueda en cuadrícula o aleatoria, la optimización bayesiana emplea un modelo sustituto para aproximar la función objetivo y guiar la búsqueda hacia regiones prometedoras del espacio de parámetros Gardner, Kusner, Xu, Weinberger, y Cunningham (2014).

Según Gardner y cols. (2014), este enfoque es especialmente útil en problemas donde cada evaluación de la función implica un alto costo computacional o de tiempo, como en la sintonización de hiperparámetros de modelos de aprendizaje automático o en el diseño de experimentos complejos.

Una extensión importante de este marco es la Optimización Bayesiana con Restricciones de Desigualdad (cBO), que incorpora la evaluación de restricciones igualmente costosas. En este escenario, no solo se busca optimizar una función objetivo, sino también satisfacer una o varias restricciones. La estrategia propuesta por Gardner y cols. combina la mejora esperada con la probabilidad de factibilidad, dando lugar a una función de adquisición restringida que prioriza regiones factibles y prometedoras.

Bases Teóricas

Dispositivos Edge.

El incremento en la demanda de los servicios y aplicaciones en las últimas décadas del uso de la Internet, ha contribuido a un fuerte aumento de los requisitos de procesamiento y almacenamiento de datos. Son diversos, en términos de los recursos que requieren las diferentes aplicaciones y, por lo tanto, a menudo invocan soluciones a medida (Doluí y Kanti Datta, 2017).

Según Medina (2019) Edge Device, en este contexto, se refiere a elementos con capacidades limitadas que tiene su propio conjunto de recursos: CPU, memoria, almacenamiento, y red. Pueden ser Smartphone, Smartglasses, smartwatches, tablets, routers, vehículos autónomos, o cualquier dispositivo de IoT con capacidad de proceso. En este escenario, Edge Computing, representa una solución para el procesamiento descentralizado de datos mediante dispositivos ubicados en el borde de la red, lo cual resulta fundamental en aplicaciones que requieren baja latencia, procesamiento en tiempo real y garantías de privacidad al evitar la transmisión de información sensible a servicios externos.

Shi, Cao, Zhang, Li, y Xu (2016) definen Edge Computing como un paradigma de computación distribuida que acerca el procesamiento y almacenamiento de datos a la fuente de generación, es decir, al “borde” de la red. Este enfoque permite reducir la latencia, reducir el uso del ancho de banda y mejorar la latencia en aplicaciones que requieren respuestas en tiempo real, como el Internet de las Cosas (IoT), la realidad aumentada, los vehículos autónomos y las ciudades inteligentes. Edge Computing se basa en dispositivos periféricos (edge devices) y nodos locales que realizan tareas de procesamiento y almacenamiento, lo que reduce la dependencia de la infraestructura centralizada de la nube.

El edge computing tambien puede ser autonomo, es decir, puede operar sin conexión a internet. Esto es especialmente relevante en aplicaciones donde la conectividad es limitada o intermitente, o donde la privacidad y seguridad de los datos son prioritarias. Al procesar los datos localmente, se minimiza la exposición a riesgos asociados con la transmisión de información a través de redes públicas.

Raspberry pi.

Según Richardson y Wallace (2016), la Raspberry Pi es un dispositivo de computación de bajo costo y alto rendimiento que ha ganado popularidad en diversos campos debido a su versatilidad y facilidad de uso. Este dispositivo, del tamaño de una tarjeta de crédito, está equipado con un procesador ARM, memoria

RAM, puertos de entrada/salida y conectividad de red, lo que lo hace una opción válida para proyectos educativos, de automatización y desarrollo de prototipos. La Raspberry Pi es capaz de ejecutar sistemas operativos basados en Linux, lo que permite a los usuarios programar y personalizar sus aplicaciones según sus necesidades (Richardson y Wallace, 2016).

Además, Upton y Halfacree (2020) destacan que la Raspberry Pi ha evolucionado significativamente desde su lanzamiento, con modelos más potentes como la Raspberry Pi 4, que ofrece mayores capacidades de procesamiento, almacenamiento y conectividad. Este modelo incluye soporte para redes Gigabit Ethernet, puertos USB 3.0 y salidas de video de alta definición, lo que lo convierte en una herramienta poderosa para aplicaciones más exigentes, como servidores domésticos, centros multimedia y sistemas embebidos avanzados (Upton y Halfacree, 2020).

Python.

Python es un lenguaje de programación poderoso, elegante y fácil de leer, diseñado para simplificar la creación de programas mediante una sintaxis clara y estructurada. Según el documento, Python destaca por su versatilidad en aplicaciones del mundo real, su enfoque en la legibilidad del código y su capacidad para integrar paradigmas como la programación orientada a objetos y funcional. Además, es software libre con una comunidad activa y una implementación estándar consolidada (Yuill y Halpin, 2006).

Tensorflow.

Según Goldsborough (2016) TensorFlow es una biblioteca de software de deep learning de código abierto desarrollada por Google que permite definir, entrenar y desplegar modelos de machine learning mediante la representación de algoritmos como grafos computacionales.

La librería TensorFlow opera construyendo un grafo computacional en el que cada nodo representa una operación (por ejemplo, una función matemática, una transformación o una capa de una red neuronal) y cada arista transporta un tensor, es decir, un arreglo multidimensional de datos. Esta arquitectura facilita diversas mejoras de rendimiento, como la eliminación de subgrafos redundantes, y permite distribuir la ejecución de la computación a lo largo de múltiples dispositivos (CPUs, GPUs, TPUs) e incluso en entornos distribuidos. De esta manera, se reduce tanto el uso de memoria como el rendimiento, haciendo posible el entrenamiento y despliegue de modelos complejos a gran escala. (Goldsborough, 2016).

NestJs.

Según Sabo (2020) Sabo (2020) NestJS es un framework para el desarrollo de aplicaciones del lado del servidor basado en Node.js, que se escribe en TypeScript. Proporciona una estructura modular y escalable mediante el uso de patrones modernos como la inyección de dependencias, controladores y módulos, facilitando la creación de aplicaciones backend mantenibles y robustas.

NestJS aprovecha la solidez de Node.js y Express.js, pero se diferencia al introducir un enfoque inspirado en Angular para la organización de la aplicación. Gracias a su arquitectura basada en módulos, cada parte de la aplicación se encapsula en unidades independientes que facilitan la reutilización y la escalabilidad. Además, el framework implementa un avanzado sistema de inyección de dependencias, lo que permite gestionar y suministrar las instancias de servicios de manera automática, reduciendo el acoplamiento entre componentes y promoviendo un diseño orientado a pruebas. Otro pilar fundamental de NestJS es su fuerte integración con TypeScript, lo cual aporta tipificación estática y facilita la detección temprana de errores durante el desarrollo. La utilización de decoradores en NestJS permite agregar metadatos a clases, métodos y propiedades, lo que habilita la implementación de características como interceptores, pipes y controladores para la validación y transformación de datos, así como para el manejo de rutas HTTP. Además, la herramienta Nest CLI agiliza la generación de nuevos proyectos y componentes, garantizando que se siga una estructura coherente en toda la aplicación, lo que resulta especialmente útil en proyectos complejos y colaborativos (Sabo, 2020).

Inteligencia artificial.

La Inteligencia Artificial (IA) se define como el estudio de agentes que perciben su entorno a través de sensores y actúan sobre él mediante actuadores, con el objetivo de maximizar su utilidad esperada. Según Russell y Norvig (2022), “ La IA es el estudio de agentes que reciben percepciones del entorno y realizan acciones. Cada uno de estos agentes implementa una función que asigna secuencias de percepción a acciones, y cubrimos diferentes formas de representar estas funciones para lograr el mejor resultado esperado.” (p. 19).

Los fundamentos de la IA se entrelazan con múltiples disciplinas. La filosofía aporta marcos éticos y lógicos, como señalan los autores: “El filósofo griego Aristóteles fue uno de los primeros en intentar codificar el “pensamiento correcto” sus silogismos proporcionaron patrones para estructuras argumentales que siempre produjeron conclusiones correctas.” (p. 21). Las matemáticas y la estadística proporcionan

herramientas para el razonamiento probabilístico: “La probabilidad rápidamente se convirtió en una parte invaluable de las ciencias cuantitativas, ayudando a lidiar con mediciones inciertas y teorías incompletas.” (p. 26). La economía contribuye con teorías de decisión y utilidad: “La teoría de la decisión, que combina la teoría de la probabilidad con la teoría de la utilidad, proporciona un marco formal y completo para las decisiones individuales tomadas en condiciones de incertidumbre.” (p. 28). La neurociencia y la psicología inspiran modelos cognitivos: “El campo interdisciplinario de la ciencia cognitiva reúne modelos informáticos de la IA y técnicas experimentales de la psicología para construir teorías precisas y comprobables de la mente humana.” (p. 21). Por último, la ingeniería y la computación permiten implementar sistemas eficientes: “La historia de la IA es también la historia del diseño de arquitecturas cada vez más sofisticadas para programas de agentes.” (p. 65).

La IA actual ha alcanzado hitos significativos en diversos dominios. De acuerdo a lo expuesto por Russell y Norvig (2022): “Los sistemas que usan IA han alcanzado o superado el rendimiento humano en ajedrez, Go, póquer, Pac-Man, Jeopardy, detección de objetos ImageNet, reconocimiento de voz y diagnóstico de retinopatía diabética.” (p. 46). En aplicaciones prácticas, destacan los vehículos autónomos: “Los vehículos de prueba de Waymo superaron la marca de 10 millones de millas recorridas en vías públicas sin sufrir accidentes graves.” (p. 47), y sistemas de diagnóstico médico: “Los algoritmos de IA ahora igualan o superan a los médicos expertos en el diagnóstico de muchas afecciones, como el cáncer metastásico y las enfermedades oftálmicas.” (p. 48). Además, herramientas como “Los sistemas de traducción automática ahora permiten la lectura de documentos en más de 100 idiomas, lo que genera cientos de miles de millones de palabras por día.” (p. 47) evidencian su impacto global. No obstante, persisten retos éticos y técnicos, como la alineación de valores humanos y la escalabilidad en entornos complejos, que definen la frontera actual de investigación.

Cadenas de Markov.

Fundamentos teóricos Las Cadenas de Markov, nombradas en honor al matemático ruso Andrei Andreevich Markov (1856–1922), son procesos estocásticos que modelan sistemas donde el futuro depende únicamente del estado presente, sin influencia directa del pasado (Matas Soberón, 2024, p. 13). Este principio, conocido como propiedad de Markov, fue inicialmente aplicado por Markov en el análisis de secuencias de vocales y consonantes en textos literarios, como Eugene Onegin de Pushkin, sentando las bases para su uso en campos como la física, biología y ciencias sociales (Matas Soberón, 2024, p. 11). Posteriormente, estas cadenas han sido utilizadas para modelar sistemas en epidemiología, teoría de colas y

dinámica de poblaciones (Bobadilla Osses, 2010, p. 5).

Una Cadena de Markov se define formalmente como un proceso estocástico X_n sobre un espacio de estados S (finito o numerable), donde la probabilidad de transición al siguiente estado depende exclusivamente del estado actual. Esta propiedad se traduce en una matriz de transición $P = (p_{ij})$, donde cada entrada p_{ij} representa la probabilidad de pasar del estado i al estado j . La matriz P es estocástica, es decir, sus filas suman 1 y sus elementos son no negativos (Matas Soberón, 2024, p. 14-15). Además, en sistemas más avanzados, se pueden considerar Cadenas de Markov de tiempo continuo, las cuales presentan una relación con la teoría de semigrupos de operadores (Bobadilla Osses, 2010, p. 6).

Las cadenas de Markov pueden clasificarse en homogéneas, cuando la matriz de transición P permanece constante en el tiempo (Matas Soberón, 2024, p. 23). Un estado es recurrente si la cadena regresa a él infinitas veces, y transitorio si eventualmente lo abandona para siempre. Esto se determina mediante la representación canónica de P , que identifica clases cerradas (conjuntos de estados que no pueden abandonarse) (Matas Soberón, 2024, p. 24-25). En el contexto de sistemas biológicos, por ejemplo, una cadena de Markov puede utilizarse para modelar la propagación de una enfermedad dentro de una población, donde los estados pueden representar niveles de infección y la recurrencia indicar posibles rebrotos (Bobadilla Osses, 2010, p. 91).

Para cadenas regulares (matrices P con todas sus entradas positivas en alguna potencia), las probabilidades convergen a un estado estacionario w , único vector estocástico que satisface $w = wP$. Este resultado se fundamenta en el teorema de Perron-Frobenius y técnicas como la descomposición de Jordan (Matas Soberón, 2024, p. 29-30). Un caso particular de convergencia ocurre en modelos de colas, donde la distribución estacionaria describe la cantidad promedio de clientes en espera en el largo plazo (Bobadilla Osses, 2010, p. 84).

Los autores en sus respectivas publicaciones ilustran las utilidades de las Cadenas de Markov en casos reales. En fútbol, se utilizan para modelizar resultados (ganar, empatar, perder) en partidos de la liga española, calculando probabilidades de equilibrio para equipos ganadores y perdedores (Matas Soberón, 2024, p. 33-41). En póquer, se emplean para el análisis de rondas de apuestas y distribución de cartas, identificando estados recurrentes (como abandonar una partida) mediante matrices de transición (Matas Soberón, 2024, p. 42-48). En Google AdWords, se aplican para predecir el comportamiento de clics en anuncios, utilizando cadenas regulares para estrategias de marketing (Matas Soberón, 2024, p. 49-53). También se usan en el modelo de Reed-Frost en epidemiología para estudiar la propagación de enfermedades infecciosas, clasificando estados en susceptibles, infectados y recuperados (Bobadilla Osses,

2010, p. 91-107). Otro ejemplo es su aplicación en cadenas de colas, modelos de atención en sistemas de servicio como supermercados o sistemas de telecomunicaciones, donde la matriz de transición describe la dinámica de llegada y atención de clientes (Bobadilla Osses, 2010, p. 57-61).

Agente.

Según Russell y Norvig (2022), un agente es cualquier entidad que percibe su entorno a través de sensores y actúa sobre él mediante actuadores. En este sentido, un agente puede ser un ser humano, un robot o un sistema de software, siempre que cumpla con esta función de percepción y acción. Por ejemplo, un agente humano posee sensores como los ojos y oídos, y actuadores como las manos y las piernas. Un agente robótico puede incluir cámaras y sensores infrarrojos como entrada, y motores como salida. Un software, en cambio, percibe datos en forma de archivos o paquetes de red y responde modificando archivos o enviando información a otros sistemas (Russell y Norvig, 2022).

El comportamiento de un agente está determinado por su función de agente, la cual define cómo actúa en función de la secuencia de percepciones que ha experimentado. En otras palabras, un agente toma decisiones basándose en su conocimiento previo y en los datos que recibe en tiempo real. Russell y Norvig (2022) también introducen el concepto de agente racional, que es aquel que elige acciones que maximizan su desempeño según un criterio predefinido. La complejidad del entorno en el que opera el agente influye directamente en su diseño y en su capacidad de tomar decisiones óptimas (Russell y Norvig, 2022).

Ciencia de datos.

Según Provost y Fawcett (2013), la ciencia de datos se enfoca en el uso de datos para tomar decisiones informadas, utilizando herramientas como el aprendizaje automático, la minería de datos y el análisis estadístico. Los autores destacan que la ciencia de datos no solo se trata de manejar grandes volúmenes de datos, sino también de entender cómo estos pueden ser utilizados para generar valor en diferentes contextos, como en el ámbito empresarial, científico o social. Por su parte, Russell y Norvig (2022) enfatizan que la ciencia de datos es una disciplina que se apoya en la inteligencia artificial y el aprendizaje automático para modelar y predecir comportamientos a partir de datos. Los autores mencionan que la ciencia de datos es fundamental en la creación de sistemas inteligentes que pueden aprender de los datos y mejorar su desempeño con el tiempo. Además, resaltan la importancia de la ética en el manejo de datos, especialmente en aplicaciones que involucran la privacidad y la seguridad de las personas. García,

Molina, y Berlanga (2018) complementan esta definición al señalar que la ciencia de datos es una disciplina que integra técnicas analíticas avanzadas, como el aprendizaje estadístico y la minería de datos, para resolver problemas complejos. Los autores destacan que la ciencia de datos no solo se limita al análisis de datos, sino que también incluye la preparación, limpieza y transformación de los datos para que puedan ser utilizados en modelos predictivos y descriptivos. Además, resaltan la importancia de la visualización de datos como una herramienta clave para comunicar los resultados de manera efectiva.

En el contexto del Sistema de Monitoreo Acústico para Identificar Sonidos y Generar Alertas de Emergencia, la ciencia de datos juega un papel fundamental en varias etapas del proyecto. En primer lugar, se utiliza para el procesamiento y análisis de señales acústicas, donde se aplican técnicas de aprendizaje automático para clasificar sonidos ambientales y detectar eventos anómalos. Como mencionan Provost y Fawcett (2013), el aprendizaje automático es una herramienta poderosa para la clasificación de datos, especialmente en aplicaciones que requieren respuestas en tiempo real, como es el caso de este sistema.

Además, la ciencia de datos es esencial en la creación y entrenamiento de modelos predictivos que permiten identificar patrones de sonidos asociados a situaciones de emergencia. Russell y Norvig (2022) destacan que los modelos de inteligencia artificial, como las redes neuronales y las cadenas de Markov, son fundamentales para la predicción y clasificación de eventos basados en datos históricos. En este proyecto, se utilizan modelos preentrenados, como YAMNet, y se adaptan para la detección de sonidos específicos, lo que demuestra la aplicación práctica de la ciencia de datos en la creación de sistemas inteligentes. Por último, García y cols. (2018) resaltan la importancia de la visualización de datos en la comunicación de resultados. En este proyecto, la ciencia de datos no solo se limita al análisis de sonidos, sino que también incluye la generación de alertas visuales y notificaciones que permiten a los usuarios entender rápidamente la situación y tomar acciones adecuadas. Esto demuestra cómo la ciencia de datos puede ser utilizada para mejorar la usabilidad y efectividad de los sistemas de monitoreo.

Técnicas predictivas para series temporales.

Según Hyndman y Athanasopoulos (2018), las series temporales son secuencias de datos ordenados en el tiempo, y su análisis implica la identificación de patrones como tendencias, estacionalidad y ciclos, para luego utilizarlos en la predicción de valores futuros. Las técnicas predictivas para series temporales son métodos estadísticos y de aprendizaje automático que permiten modelar y predecir el comportamiento futuro de datos que varían en el tiempo. Estas técnicas son fundamentales en aplicaciones donde es necesario anticipar tendencias, patrones o eventos futuros basados en datos históricos.

LSTM (Long Short-Term Memory).

Las redes neuronales recurrentes (RNN) fueron diseñadas para procesar datos secuenciales, pero sufrían el problema del desvanecimiento del gradiente, lo que dificultaba capturar dependencias de largo plazo (Heaton, 2018). Para resolverlo, Hochreiter y Schmidhuber introdujeron las LSTM, que incorporan un mecanismo de memoria a través de una celda con tres puertas: de olvido, entrada y salida. Estas puertas permiten seleccionar qué información descartar, almacenar o transmitir, logrando que el modelo aprenda patrones de secuencia de manera más robusta.

Las redes neuronales recurrentes (RNN) fueron diseñadas para procesar datos secuenciales, pero se enfrentaban a un desafío importante: la incapacidad de retener información a largo plazo debido al problema del desvanecimiento del gradiente (Heaton, 2018). Para superar esta limitación crítica, Hochreiter y Schmidhuber (1997) desarrollaron las redes LSTM. Esta arquitectura innovadora no solo procesa secuencias de manera efectiva, sino que también introduce un mecanismo de memoria que le permite recordar información relevante por largos períodos, lo que era un obstáculo insalvable para las RNN tradicionales.

La capacidad distintiva de una red LSTM radica en su celda de memoria (cell state), que actúa como una cinta transportadora que lleva información relevante a lo largo de toda la secuencia de la red. A diferencia de una neurona simple, una celda LSTM está equipada con tres “puertas” reguladoras: la puerta de olvido (forget gate), la de entrada (input gate) y la de salida (output gate). Estas puertas, controladas por funciones sigmoides, deciden qué información debe ser eliminada, qué información nueva debe ser agregada y qué información debe ser utilizada para generar la salida, respectivamente (Heaton, 2018).

La interacción entre estas puertas es lo que le da a las LSTM su poder. La puerta de olvido revisa el estado anterior y la entrada actual para determinar qué datos ya no son necesarios, limpiando la memoria. Luego, la puerta de entrada evalúa la nueva información y decide si es lo suficientemente importante para ser almacenada en la celda de memoria. Finalmente, la puerta de salida toma decisiones sobre la información contenida en la celda de memoria y la entrada actual para generar el valor de salida de la célula, permitiendo que la red utilice el conocimiento aprendido para una tarea específica (Heaton, 2018).

Gracias a este control de memoria, las LSTM han demostrado ser efectivas en una amplia gama de aplicaciones. En el procesamiento del lenguaje natural (NLP), son fundamentales para la traducción automática y la generación de texto, ya que pueden capturar las dependencias gramaticales a largo plazo. También son cruciales en la detección de anomalías en secuencias de datos, como el audio, donde pueden

aprender los patrones de sonido normales para luego identificar desviaciones que indican la presencia de una anomalía. Su éxito se debe a que superan la limitación de las RNN, permitiendo un mejor manejo de las dependencias a largo plazo (Hochreiter y Schmidhuber, 1997).

Autoencoders.

Los autoencoders son redes neuronales no supervisadas diseñadas para aprender una representación comprimida de los datos de entrada (codificación) y luego reconstruirla (decodificación). Su objetivo es minimizar el error de reconstrucción, lo que permite capturar las características más importantes de los datos. En el caso del LSTM Autoencoder, se combinan las capacidades de los autoencoders con las LSTM, resultando especialmente útiles para datos secuenciales. Este tipo de modelo también sirve para simplificar los datos y encontrar patrones importantes, lo que ayuda a tareas como predecir valores futuros, comprimir información o mejorar otros modelos de aprendizaje (Malhotra, Vig, Shroff, Agarwal, y cols., 2015).

Isolation Forest (Bosque de Aislamiento).

Isolation Forest es un método no supervisado de detección de anomalías que aísla observaciones mediante particiones aleatorias en bosques de árboles, donde las instancias atípicas requieren menos divisiones y, por tanto, muestran longitudes de camino esperadas más cortas. Su puntaje de anomalía deriva de la profundidad normalizada de aislamiento y no presupone distribuciones paramétricas ni datos etiquetados, lo que favorece su uso en flujos en línea y alta dimensionalidad (Aggarwal, 2016).

A diferencia de la mayoría de los algoritmos de detección de anomalías, que intentan construir un perfil complejo de los datos “normales” para luego identificar lo que no encaja, Isolation Forest (IF) se basa en un principio mucho más directo. La premisa fundamental es que las anomalías son, por definición, “pocas y diferentes”, lo que las hace más susceptibles a ser aisladas que los puntos de datos normales (Liu, Ting, y Zhou, 2012).

El algoritmo funciona construyendo un conjunto de árboles de decisión aleatorios, conocidos como “árboles de aislamiento” (isolation trees). Para cada árbol, el conjunto de datos se partitiona recursivamente seleccionando un atributo al azar y luego un valor de división aleatorio entre el valor máximo y mínimo de ese atributo. Este proceso de división aleatoria se repite hasta que cada punto de datos queda aislado en un nodo hoja del árbol. La idea central es que los puntos anómalos, al ser diferentes y estar más alejados de las concentraciones de datos normales, requerirán menos particiones para ser aislados (Liu y cols., 2012).

La “anomalía” de un punto se mide calculando la longitud del camino (path length) desde la raíz del árbol hasta el nodo hoja que lo contiene. Los puntos normales, que se encuentran en regiones densas, necesitarán muchos más cortes para ser aislados, resultando en caminos más largos. En contraste, las anomalías, que se encuentran en zonas de baja densidad, serán aisladas con muy pocos cortes, resultando en caminos muy cortos. El puntaje final de anomalía para cada punto se calcula promediando la longitud de su camino a través de todos los árboles del bosque, lo que hace que el resultado sea robusto y fiable (Liu y cols., 2012).

Gracias a su simplicidad y a no depender de cálculos de distancia o densidad, Isolation Forest es extremadamente rápido y tiene un bajo consumo de memoria. Estas características lo convierten en un candidato para la detección de anomalías en tiempo real sobre grandes volúmenes de datos, y es particularmente adecuado para ser implementado en dispositivos de borde (Edge) con recursos computacionales limitados, como es el caso de este proyecto.

Yamnet.

Desarrollado por Google, YAMNet es un modelo preentrenado disponible en formato TFLite, lo que lo hace ideal para su implementación en sistemas embebidos o de bajo consumo computacional Google (s.f.). Este modelo fue descargado desde la plataforma Kaggle, en su página oficial. YAMNet está entrenado sobre el conjunto de datos AudioSet, que contiene más de 2 millones de segmentos de audio etiquetados con 521 clases de eventos sonoros, como gritos, ladridos, música, ruidos ambientales, entre otros Google (s.f.). AudioSet es un dataset a gran escala que busca proporcionar una cobertura integral de los sonidos del mundo real Gemmeke y cols. (2017a). Esta amplia cobertura de clases convierte a YAMNet en una herramienta versátil para la detección de eventos críticos.

La arquitectura de YAMNet se basa en MobileNet v1, una red neuronal convolucional (CNN) diseñada específicamente para aplicaciones móviles y embebidas Google (s.f.). Esta arquitectura, que utiliza convoluciones separables en profundidad (depth-wise separable convolutions) para reducir la complejidad computacional Howard y cols. (2017), permite ejecutar el modelo en dispositivos con limitaciones de hardware sin sacrificar el rendimiento. El modelo acepta como entrada una forma de onda de audio en formato mono, muestrada a 16 kHz, y normalizada en el rango de $[-1,0, +1,0]$. Internamente, YAMNet divide la forma de onda en ventanas de 0.96 segundos con un salto (hop) de 0.48 segundos Google (s.f.), lo que permite analizar el audio en tiempo real. Cada ventana se procesa de manera independiente, generando tres tipos de salidas: scores, que contienen las puntuaciones de predicción para

cada una de las 521 clases; y embeddings, que representan características de alto nivel extraídas del audio.

Redis Pub/Sub.

Según Carlson (2013), Redis ofrece un sistema de publicación-suscripción (Pub/Sub) que permite a los clientes enviar mensajes a canales sin necesidad de conocer a los receptores. Esta arquitectura desacoplada es útil para sistemas distribuidos, notificaciones en tiempo real y flujos de eventos. Los clientes pueden suscribirse a uno o varios canales y recibir mensajes en tiempo real, sin almacenamiento ni persistencia. Carlson (2013) destaca que este mecanismo se caracteriza por su baja latencia y simplicidad: los mensajes no se almacenan, se entregan en mejor esfuerzo y se pierden si no hay suscriptores activos, lo que lo hace idóneo para eventos efímeros y señales transitorias.

Embedding computing (Computación Embebida).

La Computación Embebida se define como un sistema computacional diseñado y programado para ejecutar una o un conjunto limitado de funciones dedicadas, formando un componente integral e inseparable dentro de un sistema mecánico o eléctrico más amplio. Según Wolf (2002), quien establece que un sistema embebido es “quier computadora que sea un componente en un sistema más grande y que dependa de su propio microprocesador”. Esta arquitectura se distingue radicalmente de las computadoras de propósito general por su especialización, estando optimizada para operar bajo restricciones críticas de tamaño, costo, consumo energético y, fundamentalmente, de tiempo real, un atributo donde la corrección de su operación no solo depende de la precisión lógica, sino también del estricto cumplimiento de los plazos temporales para interactuar de forma efectiva con el entorno físico a través de sensores y actuadores.

Bases Legales

El desarrollo del Sistema de Monitoreo Acústico se fundamenta en el cumplimiento de las normativas venezolanas vigentes relacionadas con la privacidad, protección de datos y seguridad informática. A continuación, se detallan los instrumentos legales aplicables:

1. Constitución de la República Bolivariana de Venezuela (1999)

- Artículo 60: Garantiza el derecho a la protección de la vida privada, intimidad, honor, propia

imagen, confidencialidad y reputación.

- Artículo 28: Establece el derecho a la protección de datos personales.

2. Ley Especial contra Delitos Informáticos (2001)

- Artículo 6: Prohíbe el acceso no autorizado a sistemas informáticos.
- Artículo 20: Sanciona la violación de la privacidad mediante la interceptación de comunicaciones.

3. Ley Orgánica de Telecomunicaciones (2010)

- Artículo 3: Promueve el uso ético de las telecomunicaciones.

Capítulo III. Marco Metodológico

Este capítulo describe el conjunto de estrategias, técnicas y procedimientos empleados para el desarrollo del Sistema de Monitoreo Acústico para Identificar Sonidos y Generar Alertas de Emergencia. En él se detalla tanto el enfoque investigativo como las herramientas utilizadas para la recolección de datos y, fundamentalmente, la metodología de desarrollo adoptada, basada en el modelo espiral.

Tipo de Investigación

El presente trabajo constituye una investigación de tipo proyectiva. De acuerdo con Hurtado (2000), la investigación proyectiva tiene como objetivo diseñar o crear propuestas dirigidas a resolver determinadas situaciones. Los proyectos de arquitectura e ingeniería, el diseño de maquinarias, la creación de programas de intervención social, el diseño de programas de estudio, los inventos, la elaboración de programas informáticos, entre otros, siempre que estén sustentados en un proceso de investigación, son ejemplos de investigación proyectiva. Este tipo de investigación potencia el desarrollo tecnológico (p. 325).

Basado en esta definición, el desarrollo de un sistema de monitoreo acústico para identificar sonidos y generar alertas de emergencia se adapta a este tipo de investigación. El sistema propuesto en este trabajo se fundamenta en un proceso de investigación que incluye la revisión documental de conceptos como edge computing, el uso de modelos pre-entrenados para la clasificación de audio y el análisis de secuencias temporales para la detección de anomalías, así como el diseño e implementación de un prototipo funcional. Esto cumple con los requisitos de una investigación proyectiva, ya que no solo se analiza y diagnostica el problema, sino que también se desarrolla una solución tecnológica aplicable en entornos reales.

Metodología de Desarrollo Utilizada

El desarrollo de un sistema de detección de monitoreo acústico que envie alertas de emergencia presenta desafíos técnicos inherentes que dificultan la definición precisa de los requisitos de implementación desde el inicio. Entre estos desafíos se encuentran, la incertidumbre sobre qué técnicas de procesamiento de audio y algoritmos de aprendizaje automático serían más efectivos para los escenarios identificados, el

desconocimiento sobre las capacidades reales de los dispositivos embebidos para ejecutar procesamiento de señales y algoritmos de IA en tiempo real y la necesidad de validar el desempeño de diferentes enfoques técnicos antes de comprometer la arquitectura completa. Estos factores hacían imprescindible un enfoque iterativo con validación experimental en cada etapa del desarrollo.

Metodologías	Flexible	Evolutivo	Gestión de riesgos	Iteración y mejora continua
Espiral	Sí	Sí	Sí	Sí
Cascada	No	No	No	No
Incremental	No	Sí	No	Sí
Scrum	Sí	Sí	No	Sí

Tabla 1: Cuadro Comparativo de metodologías

Dada esta incertidumbre técnica, se necesitaba una metodología flexible y evolutiva, capaz de gestionar activamente los riesgos de implementación y permitir el refinamiento continuo de la solución mediante ciclos de retroalimentación. Para seleccionar la metodología más adecuada, se compararon diferentes enfoques en función de cuatro características clave: flexibilidad, capacidad evolutiva, gestión de riesgos e iteración con mejora continua, como se muestra en la Tabla 1.

Se optó por el Modelo Espiral debido a su énfasis explícito en el análisis de riesgos como una fase formal y estructurada en cada ciclo. Dada la naturaleza exploratoria del proyecto y los múltiples riesgos técnicos identificados, esta característica fue considerada fundamental para mitigar de manera proactiva cualquier desafío que pudiera surgir durante la implementación.

El Modelo Espiral, propuesto por Boehm (1988), es un enfoque iterativo que combina elementos del desarrollo incremental y el prototipado, permitiendo gestionar riesgos y adaptar el sistema a medida que avanza el proyecto. Este modelo ha sido ampliamente reconocido por su capacidad para gestionar el desarrollo de sistemas complejos, ya que combina la naturaleza iterativa del prototipado con la estructura sistemática del modelo en cascada (Pressman, 2010, p. 39). En cada iteración se pueden realizar ajustes en el plan del proyecto, permitiendo adaptar el software a las necesidades emergentes y reducir riesgos antes de que se conviertan en problemas críticos (Pressman, 2010, p. 40).

Según Boehm (1988), el Modelo Espiral se caracteriza por su estructura cíclica, en la que cada iteración incluye cuatro fases principales:

1. **Determinación de objetivos.** Inicialmente se definen los objetivos específicos de la iteración, identificando las limitaciones del proceso y del sistema de software. Se evalúan alternativas y se especifican condiciones como lenguaje, entornos, entre otros.
2. **Análisis de riesgos.** Se identifican y evalúan los riesgos potenciales. Se definen acciones para reducir los riesgos identificados y se evalúan alternativas existentes partiendo de prototipos, simulaciones y softwares de análisis. En este ciclo, existen varios prototipos como plantillas de diseño o componentes funcionales.
3. **Desarrollo.** Los prototipos se amplían y se añaden funcionalidades. El código es escrito, probado y migrado a un entorno de prueba varias veces hasta que pueda ser implementado correctamente.
4. **Planificación.** Al final de cada iteración se procede a planificar la siguiente, de ser necesaria. Si se producen errores, se buscan soluciones, y si se consiguen mejores alternativas, se implementa en el siguiente ciclo.

Como señala Boehm (1988), “el Modelo Espiral es particularmente adecuado para proyectos con altos niveles de incertidumbre y requisitos cambiantes, ya que permite incorporar retroalimentación continua y adaptar el diseño en función de los resultados obtenidos”. En este trabajo, esta metodología permitió gestionar los riesgos técnicos y asegurar que el sistema cumpliera con los objetivos planteados. La Figura 1 ilustra el modelo de desarrollo en espiral propuesto por Boehm (1988).

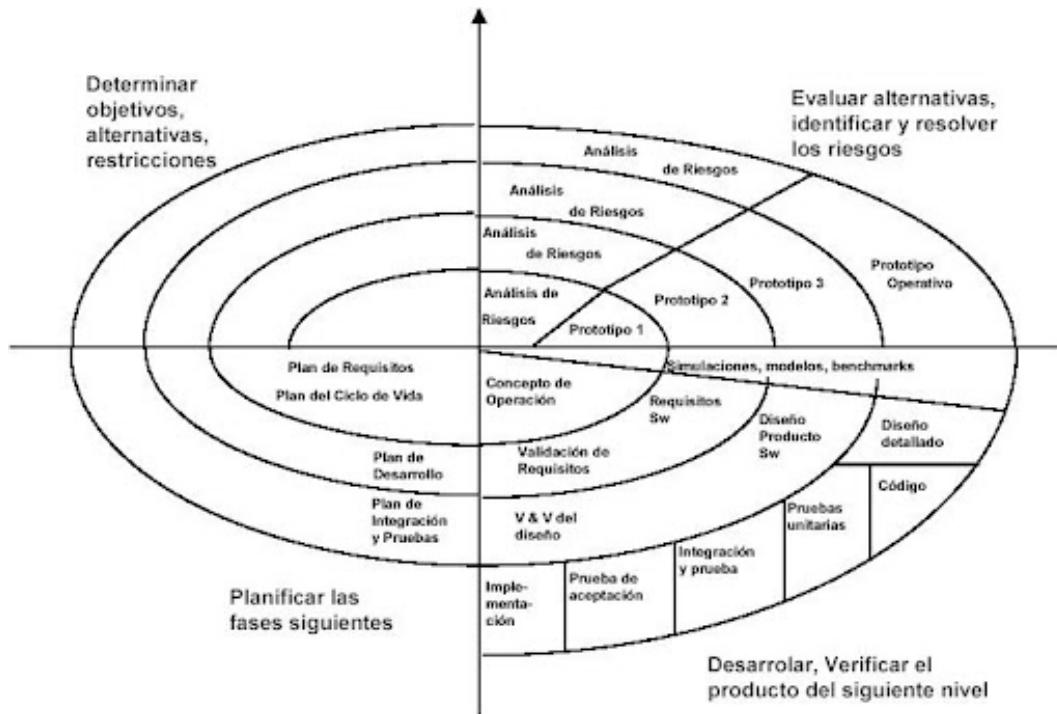


Figura 1: Modelo de desarrollo en espiral propuesto por Boehm (1988).

Capítulo IV. Desarrollo y Resultados

En este capítulo se detalla el proceso de construcción del sistema de monitoreo acústico, siguiendo la metodología y su relación con los objetivos específicos. A continuación, se presentan las actividades y resultados obtenidos en cada iteración del proyecto.

Análisis inicial del sistema de monitoreo acústico para generar alertas en casos de emergencia.

Desde el planteamiento del proyecto se identificó como un problema social relevante la detección de situaciones de emergencia en el hogar, especialmente para personas más expuestas a quedar desatendidas ante situaciones de vulnerabilidad, como adultos mayores que viven solos o personas con condiciones médicas que limitan su movilidad. Estudios como el de (Bezold, Krell-Roesch, Eckert, Jekauc, y Woll, 2021), destacan la prevalencia de caídas y emergencias médicas en estos grupos. Esta realidad evidencio la necesidad de un sistema de monitoreo que funcione de manera autónoma y discreta, sin depender de la intervención del usuario debido a su posible incapacidad para interactuar con un dispositivo durante una emergencia.

Para el análisis inicial del sistema, se comenzó con la revisión de literatura científica y técnica relacionada con la identificación y caracterización de eventos acústicos. Se prestó especial atención a estudios que abordaran la implementación de sistemas en dispositivos de Edge Computing.

Esta revisión incluyó estudios sobre el procesamiento de señales de audio y modelos de aprendizaje automático aplicados al reconocimiento de sonidos. Se encontró que era posible representar una señal de audio gráficamente como un espectrograma, que es una representación visual del sonido en función del tiempo y la frecuencia, donde la intensidad se codifica con color o brillo. Esto significa que la información necesaria para distinguir eventos sonoros, que se encuentra en las complejas estructuras de frecuencias del audio, podía ser interpretada mediante técnicas de análisis de imágenes.

Esto sugirió que, para una caracterización significativa del sonido, la estrategia de solución debía enfocarse en la Clasificación de Eventos Acústicos. En este contexto, la Inteligencia Artificial,

particularmente las Redes Neuronales Convolucionales (CNN), se presentó como el enfoque más adecuado para procesar y etiquetar la información compleja contenida en la estructura de frecuencias (el espectrograma). Este enfoque se ve respaldado por la literatura; por ejemplo, investigaciones en el diagnóstico de fallos de maquinaria han demostrado la efectividad de utilizar representaciones spectrales, como el Mel Spectrogram y el Scalogram, para la detección automática de anomalías acústicas mediante arquitecturas de Deep Learning (Tran y Lundgren, 2020).

Sabiendo que la clasificación de eventos acústicos conforma un componente necesario para la solución se identificó una limitante, la clasificación por sí sola no resolvía el problema, debido a la alta variabilidad del entorno sonoro doméstico. No es lo mismo caracterizar sonidos en un entorno donde el comportamiento de las señales de audio es estable y predecible (como en una máquina), que en un hogar donde los sonidos son altamente dinámicos y contextuales. Es decir, la clasificación de eventos aislados no proporciona suficiente contexto para determinar si un evento es crítico o no. Por ejemplo, escuchar un grito sin contexto no permite determinar si es una situación de emergencia o una expresión normal de alegría o sorpresa. En otras palabras, la clasificación de eventos aislados es insuficiente para detectar situaciones críticas, ya que carece del contexto necesario para interpretar adecuadamente los eventos sonoros.

Ante esta limitante y asumiendo que las emergencias son eventos poco frecuentes y atípicos dentro del patrón de actividad sonora normal del hogar, se consideró la necesidad de analizar el comportamiento temporal de los eventos acústicos. Esto implicaba no solo clasificar los sonidos, sino también entender cómo estos eventos se distribuían a lo largo del tiempo para identificar patrones normales y detectar desviaciones significativas que pudieran indicar una emergencia.

Para abordar la necesidad del análisis de secuencias temporales, se revisaron estudios previos sobre predicción de eventos basados en series temporales. El uso de modelos predictivos como ARIMA, SARIMA o Prophet es una práctica establecida en la literatura para el desarrollo de sistemas de alerta temprana mediante el análisis y la predicción de patrones temporales Mora y Losada (2023). Estos modelos se perfilaban como una estrategia para establecer umbrales de predicción sobre la secuencia de eventos acústicos. El objetivo era determinar qué tan “típica” resultaba la ocurrencia en un momento dado, sirviendo así como un detector de patrones de comportamiento anómalo.

En función del análisis realizado, se procedió a definir los requerimientos del sistema de monitoreo acústico para la detección de emergencias en el hogar. La verificación de los mismos se encuentra en la tabla

Requerimientos Funcionales.

1. El sistema debe capturar el audio ambiental de forma continua a través de los micrófonos.
2. El sistema debe procesar el audio capturado para su caracterización.
3. El sistema debe caracterizar el perfil de la actividad acústica “típica” del entorno, basado en la estacionalidad de los eventos sonoros clasificados.
4. El sistema debe procesar y comparar la actividad sensada de manera inmediata con el perfil de normalidad para detectar patrones anómalos con la mínima latencia posible.
5. El sistema debe enviar notificaciones de emergencia a una lista de contactos predefinidos si se detecta una anomalía o grito de auxilio.

Requerimientos no funcionales.

1. El sistema debe procesar y analizar los datos de audio en el borde (edge) sin almacenar las grabaciones de audio bruto ni información personal identificable.
2. El sistema debe ser capaz de integrar de manera flexible nuevos dispositivos de procesamiento (hardware) y/o de captura (micrófonos).
3. El sistema debe contar con una característica de inicio automático después de cualquier corte de energía o reinicio.
4. El sistema debe tener un diseño modular que simplifique y agilice las tareas de mantenimiento y actualización de sus componentes.

Procedimiento Metodológico

Primera Iteracion.

El objetivo principal de la primera iteración fue validar la hipótesis de que se podían detectar anomalías en el comportamiento sonoro del entorno mediante el análisis de series temporales de intensidad. La estrategia consistió en utilizar un modelo estadístico para predecir el comportamiento sonoro ”normal”, de modo que cualquier diferencia significativa entre el valor predicho y el valor real observado fuera

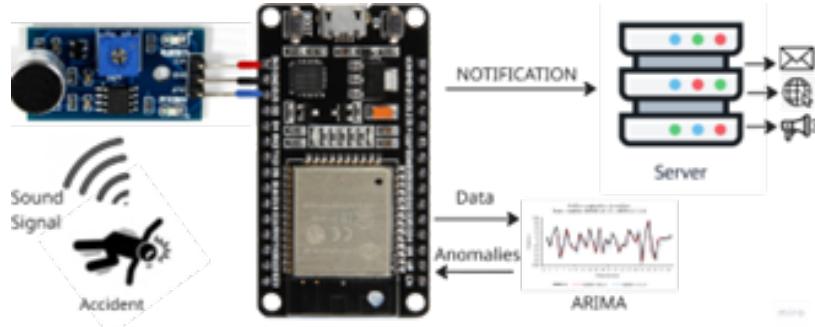


Figura 2: Diagrama de arquitectura de sistema con ESP32 y detección de intensidad de sonido

clasificada como una anomalía. El principal riesgo identificado en esta etapa fue la capacidad del modelo estadístico para funcionar adecuadamente en un entorno tan dinámico y variable como el doméstico.

Los componentes con los que contó esta iteración fueron un microcontrolador ESP32 y un micrófono que capturaba la intensidad de sonido. Véase en la figura 2. Esta serie temporal de niveles de intensidad sonora fue procesada por un modelo ARIMA (AutoRegressive Integrated Moving Average) para predecir el comportamiento esperado. Si la diferencia entre el valor predicho y el valor real superaba un umbral predefinido, se generaba una alerta.

El diseño del prototipo para probar este enfoque se inspiró en metodologías documentadas para sistemas IoT (Luis-García y Gómez, 2024) y consistió en un dispositivo de captura basado en un microcontrolador ESP32, que transmitía los datos a un servidor local (Raspberry Pi 4) para su análisis con el modelo ARIMA. Sin embargo, durante la fase de evaluación, este enfoque demostró ser fundamentalmente inviable. El problema de fondo no fue solo que el modelo ARIMA tuviera limitaciones funcionales, sino que su propio proceso de parametrización resultó ser incompatible con la naturaleza del problema de investigación.

La parametrización de ARIMA (basada en los parámetros p, d, q) busca definir una estructura de dependencia y estacionalidad fija en los datos. En un entorno doméstico, la estacionalidad de los eventos acústicos es altamente variable y depende de múltiples factores contextuales, como la hora del día, el día de la semana, o eventos imprevistos. Esta variabilidad hace que sea extremadamente difícil definir parámetros fijos que puedan capturar adecuadamente los patrones normales de comportamiento sonoro para cualquier entorno, confirmando el riesgo inicial sobre la inestabilidad.

La principal limitación funcional de esta iteración fue la incapacidad de distinguir entre diferentes tipos de anomalías, ya que el modelo solo identificaba desviaciones en la intensidad del sonido sin considerar su naturaleza o contexto (e.g., una anomalía de volumen alto podía ser tanto una alarma como un evento

crítico).

Con estas limitaciones identificadas, se decidió avanzar hacia una segunda iteración que abordara estos desafíos.

Segunda Iteracion.

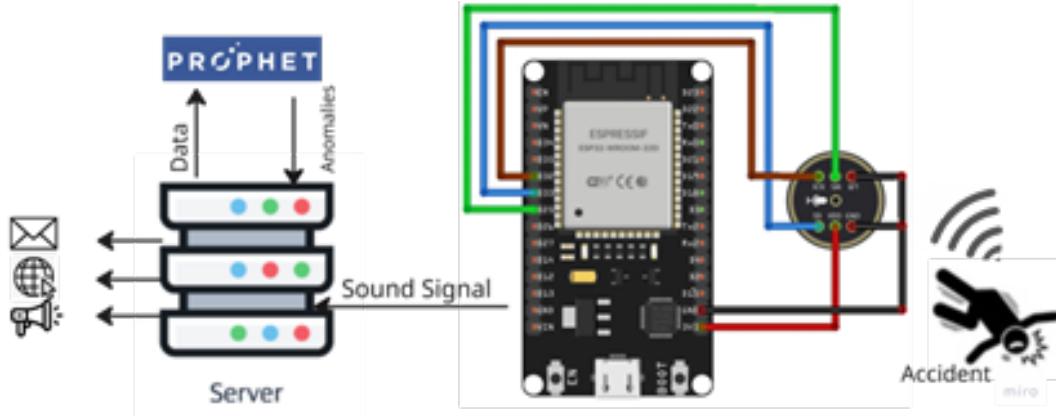


Figura 3: Diagrama de arquitectura de sistema con ESP32, micrófono de señales de audio y prophet

En la Segunda Iteración, el objetivo fue superar la rigidez del modelo ARIMA, que resultó incompatible con la dinámica del entorno doméstico, manteniendo el enfoque de análisis de series temporales. La principal limitación, que se convirtió en el riesgo metodológico de esta etapa, radicaba en la pregunta de si la naturaleza dinámica y no estacionaria de los eventos domésticos podría ser modelada adecuadamente por cualquier librería de pronóstico que requiriera patrones estables. Se mantuvo la arquitectura de hardware de Edge Computing basada en el microcontrolador ESP32 y se aplicaron mejoras en la captura de datos. Se integraron micrófonos omnidireccionales INMP441 para asegurar una captura de señales acústicas de alta fidelidad, véase en la figura 3. A nivel de software, se reemplazó el modelo ARIMA por Prophet, una librería de pronóstico de series temporales desarrollada por Facebook, elegida por su capacidad para manejar estacionalidad y tendencias complejas, buscando mayor flexibilidad para modelar el entorno.

El modelo Prophet logró ajustarse eficazmente a las variaciones periódicas del sonido ambiental, como los patrones de ruido diurnos y nocturnos. Sin embargo, durante las pruebas se identificó una limitación crítica, confirmando el riesgo inicial asociado al pronóstico: el mecanismo interno del modelo para suavizar datos e identificar tendencias a largo plazo provocaba que se descartaran los eventos de corta duración y alta amplitud de interés. Sonidos instantáneos como un golpe, un grito o la rotura de un cristal

eran filtrados y no generaban alertas, ya que no formaban parte de un patrón estacional o de una tendencia predecible. Esta observación llevó a la conclusión definitiva de que el enfoque de pronóstico de series temporales, independientemente del modelo (ARIMA o Prophet), no era el adecuado para el objetivo del proyecto. El problema no era predecir la evolución del nivel de ruido, sino la detección de eventos atípicos e instantáneos basados en el contenido acústico. Por lo tanto, el problema se redefinió como uno de detección de anomalías en tiempo real basada en la caracterización de la señal. Para implementar esta nueva estrategia, se propuso un enfoque basado en una arquitectura de redes neuronales (Deep Learning).

La Planificación de la siguiente iteración se centró en la implementación de un modelo de clasificación acústica para caracterizar los eventos sonoros, seguido de un análisis temporal de las secuencias de eventos para detectar anomalías basadas en patrones de comportamiento.

Tercera iteración.

El objetivo de esta tercera iteración fue abordar la necesidad de clasificar los tipos de sonidos detectados, limitación identificada desde la primera iteración. Inicialmente, la estrategia se centró en el desarrollo de un modelo de clasificación personalizado a partir de un conjunto de datos propio. El plan consistía en capturar y etiquetar una colección de sonidos de emergencia para entrenar una red neuronal diseñada específicamente para los escenarios de interés del proyecto. El principal riesgo metodológico de esta etapa fue la viabilidad de la adquisición de datos. Se asumió el riesgo de que la construcción de un dataset lo suficientemente vasto y diverso para garantizar la robustez y generalización del clasificador resultara inviable para los recursos del proyecto. A pesar de este riesgo, el diseño de hardware de Edge Computing se mantuvo para el desarrollo, enfocado en el procesamiento local de las señales.

Sin embargo, a medida que avanzaba el desarrollo, la investigación sobre la robustez de los clasificadores acústicos reveló la verdadera magnitud del desafío. Se hizo evidente que para que un modelo pudiera generalizar y operar de manera fiable, necesitaría ser entrenado con un volumen de datos masivo. La literatura académica confirma que la efectividad de los modelos de aprendizaje profundo para audio depende directamente de la escala y diversidad de los datos, requiriendo a menudo millones de ejemplos para capturar la enorme variabilidad de los sonidos del mundo real (Gemmeke y cols., 2017b). La tarea de construir un dataset de esa magnitud, confirmando el riesgo inicial, era logísticamente inviable para el proyecto.

Fue precisamente durante esta etapa de investigación que se identificó YAMNet, una arquitectura de

clasificación acústica ya pre-entrenada por Google. Este modelo fue entrenado sobre el corpus AudioSet, un extenso conjunto de datos que, como describen Gemmeke y cols. (2017b), contiene más de dos millones de clips de audio de 10 segundos etiquetados a partir de una ontología de 527 clases de eventos sonoros, véase el anexo C. El descubrimiento de YAMNet ofreció una solución directa al problema de la adquisición de datos a gran escala que se había identificado como el principal obstáculo. Ante esta situación, continuar con el desarrollo de un modelo propio habría significado un esfuerzo redundante. Por lo tanto, se tomó la decisión de descartar el enfoque personalizado y pivotar hacia la integración de YAMNet.

En el anexo C una figura ilustra el volumen y diversidad del dataset AudioSet utilizado para entrenar YAMNet.

Esta medida representa una aplicación práctica del paradigma de aprendizaje por transferencia (transfer learning), que permite aprovechar el conocimiento de un modelo ya entrenado en un gran dataset para resolver un problema específico, tal como analizan Pons Puig y cols. (2019) en su trabajo sobre el impacto del tamaño del dataset en audio. Este pivote tecnológico permitió enfocar los recursos restantes del proyecto en la implementación del sistema final sobre una base tecnológica robusta y ya validada, y superar la limitación de la falta de contexto que afectó a las iteraciones basadas en series temporales.

Cuarta Iteración.

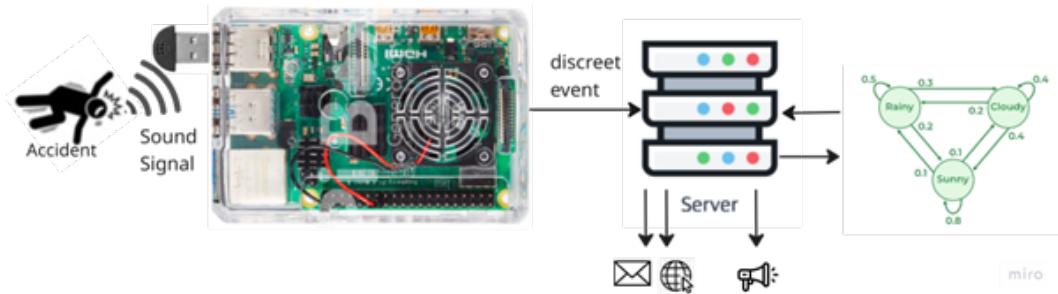


Figura 4: Diagrama de arquitectura de sistema con Raspberry Pi, Yamnet y Cadenas de Markov

En esta cuarta iteración del prototipo, se migró la lógica de procesamiento desde el ESP32 hacia una Raspberry Pi, con mayor capacidad computacional, véase en la figura 23. Se integraron micrófonos digitales de rango completo, capaces de capturar la señal de audio completa, lo que permitió acceder a un contenido espectral completo. Véase en la figura 4.

Con esta infraestructura, se incorporó YAMNet, un modelo de clasificación de audio en tiempo real basado en redes neuronales convolucionales entrenado con el dataset AudioSet de Google. YAMNet permite

detectar y clasificar una gran variedad de eventos acústicos o (más de 500 categorías) como gritos, portazos, pasos, objetos cayendo, entre otros. Esta capacidad permitió dar un salto importante en la interpretación contextual del entorno sonoro.

No obstante, Si bien YAMNet ofrecía una detección puntual efectiva de eventos sonoros, surgió la necesidad de dotar a dichos eventos de un significado secuencial y contextual. Para atender esta limitación, se optó por integrar un módulo de modelado probabilístico basado en cadenas de Márkov, con el fin de analizar la secuencia temporal de eventos y determinar si un patrón resultaba coherente con el comportamiento normal del entorno o, por el contrario, representaba una posible anomalía.

La decisión de emplear cadenas de Márkov estuvo fundamentada en dos aspectos principales: por un lado, en la existencia de estudios previos donde este tipo de modelos han sido utilizados para la detección de anomalías en secuencias temporales, destacando por su capacidad de representar probabilísticamente las transiciones entre estados inmediatos. En particular, investigaciones como la de Boldt, Borg, Ickin, y Gustafsson (2020), muestran cómo las cadenas de Márkov permiten identificar patrones irregulares en secuencias de eventos al modelar las transiciones esperadas y comparar estas con la realidad observada. Por otro lado, la elección también estuvo motivada por la recomendación del profesor de estadística Omar Castro, quien sugirió explorar este enfoque como un primer acercamiento al modelado secuencial de los eventos detectados.

No obstante, también surgieron limitaciones importantes. Si bien éramos capaces de detectar situaciones potencialmente peligrosas a partir de categorías específicas identificadas por YAMNet, el intento de modelar secuencias completas de eventos mediante cadenas de Márkov resultó ineficaz, ya que se quedaba saltando en ciclos infinitos de cambios de estado, por ejemplo (luego de Habla el siguiente evento mas probable era Habla). Estas cadenas se limitan a representar transiciones entre estados inmediatos, es decir, predicen solo el evento más probable siguiente, sin tener en cuenta el historial completo de eventos previos.

Intentar forzar el modelado de secuencias más largas con este enfoque llevó a comportamientos problemáticos, como ciclos repetitivos sin resolución lógica, o secuencias que carecían de coherencia contextual, especialmente cuando el espacio de eventos posibles crecía. Esta rigidez estructural impidió representar adecuadamente escenarios más complejos o ambiguos, donde la interpretación depende no solo del último evento detectado, sino de una serie de interacciones acústicas encadenadas en el tiempo.

Este hallazgo marcó un punto de inflexión en el desarrollo del sistema, motivando la exploración de enfoques más robustos y dinámicos, capaces de capturar dependencias de largo plazo en secuencias

acústicas. En particular, se consideraron modelos como LSTM (Long Short-Term Memory) y transformers, que permiten aprender patrones secuenciales con memoria y contexto más amplio.

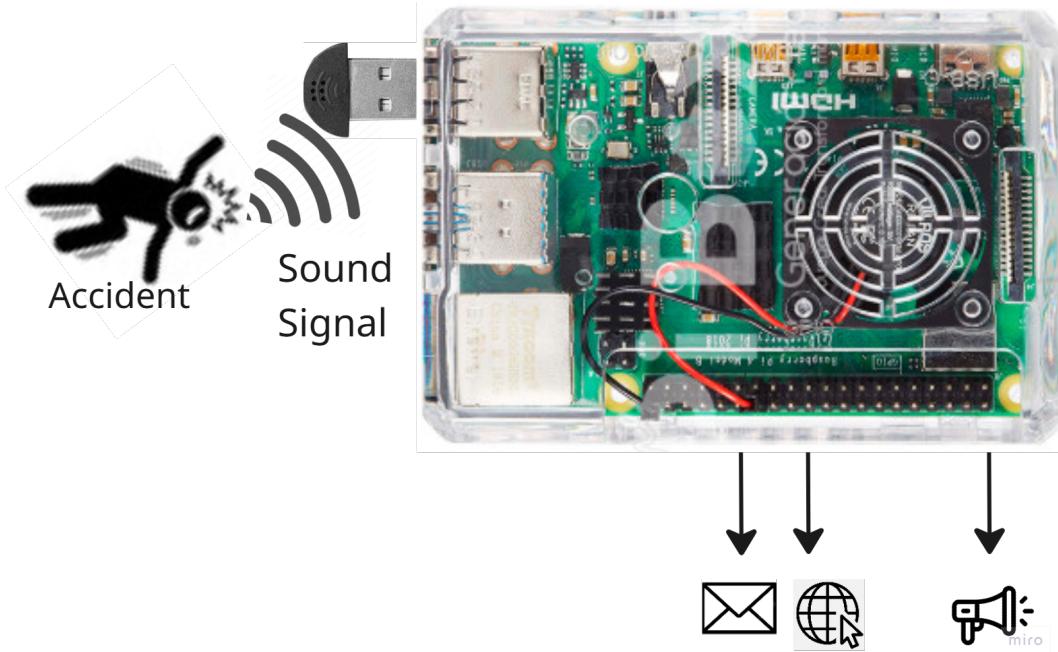


Figura 5: Diagrama de arquitectura de sistema con Raspberry Pi Autonomo, Yamnet y Modelos de IA para detección de anomalías

Quinta Iteración.

En esta quinta iteración, nos enfocamos en el desarrollo de una solución más robusta para la detección de anomalías, como respuesta a las limitaciones encontradas con las cadenas de Markov. Para ello, basamos nuestro enfoque en el trabajo de Reis y Serôdio (2025), cuya investigación se centra en arquitecturas de Inteligencia Artificial en el borde (Edge AI) para el monitoreo en tiempo real. Específicamente, su estudio valida el uso de dos tecnologías que adoptamos: Isolation Forest (IF), un modelo para la detección de eventos anomalos basado en árboles, y un (LSTM) Autoencoder, una red neuronal para el análisis de patrones secuenciales complejos. En la figura 5 se ilustra la arquitectura del sistema implementado en esta iteración.

Para la implementación de estos modelos en nuestro sistema, se replicaron las configuraciones de hiperparámetros validadas en dicho estudio para asegurar el rendimiento en la Raspberry Pi.

En el caso del Isolation Forest, adoptamos los hiperparámetros del estudio, configurando el modelo con 100 árboles ($n_{estimators}$) y un parámetro de contaminación del 3 %. Este último valor asume una

proporción esperada de anomalías en nuestros datos, alineándose con los escenarios simulados en la investigación de referencia, Véase en la figura 24

El segundo modelo implementado es un LSTM Autoencoder, una arquitectura de red neuronal no supervisada especialmente eficaz para la detección de anomalías en datos secuenciales (Malhotra y cols., 2015). Su objetivo es aprender una representación comprimida de los patrones “normales” presentes en los datos de entrenamiento para luego identificar desviaciones significativas. De igual manera, se adoptaron los hiperparámetros recomendados por Reis y Serôdio (2025), que incluyen un tamaño de lote de 32 y 50 épocas de entrenamiento. La arquitectura específica del modelo consta de 3 capas LSTM en el codificador y 3 capas LSTM en el decodificador, con tamaños de capa decrecientes de 32, 16 y 8 unidades respectivamente. y activación de ReLU para ajustar la tasa de aprendizaje y evitar el sobreajuste. Para el desarrollo y aplicación del modelo, se adoptó un enfoque metodológico inspirado en el trabajo de Reis y Serôdio (2025), el cual se estructuró en un proceso de tres etapas: preparación de los datos, construcción y entrenamiento de la arquitectura, y definición del mecanismo de detección.

1. Preparación de los Datos: Para que nuestro modelo LSTM pudiera interpretar los datos de los eventos, fue necesario “traducirlos” a un formato numérico que la red neuronal pudiera procesar. Este paso es crucial. Si bien el modelo puede ser entrenado con fechas o etiquetas de texto planas, este trabaja mejor cuando los datos se representan con números, y mientras más sentido y consistencia tengan esos números mediante un pre-procesamiento adecuado, mejor podrá aprender. Por ejemplo, si convertimos los días de la semana a números (domingo=0, lunes=1, ..., sábado=6), el modelo interpretaría que el sábado (6) y el domingo (0) son valores muy distantes, cuando en realidad son adyacentes. Este tipo de malentendido impide que la red aprenda patrones que ocurren en la transición del fin de semana. Para resolver este problema, el enfoque consiste en representar las variables temporales de una manera que refleje su naturaleza cíclica. En lugar de ver el tiempo como una línea recta, lo tratamos como un círculo, similar a las manecillas de un reloj. Esto se logra mediante funciones matemáticas (seno y coseno) que asignan a cada instante una coordenada única en un círculo. De esta forma: El sábado queda matemáticamente “cerca” del domingo. Las 23:59 quedan justo al lado de las 00:00. Véase el Apéndice A. Este preprocesamiento es fundamental, pues permite que el modelo LSTM capture patrones de comportamiento continuos y lógicos, como los que ocurren durante la madrugada o en el cambio de un día para otro.

De manera similar, el modelo puede procesar etiquetas de texto como “uso de microondas” o “sensor de movimiento”. Pero le facilitaría el aprendizaje si estas etiquetas se convierten en

números. Un error común sería asignarles un número a cada una (ej: microondas=1, sensor=2), ya que esto crearía una falsa relación matemática (como si el “sensor” fuera el doble del “microondas”). La solución es una técnica llamada One-Hot Encoding. Funciona como un panel de interruptores, donde cada evento posible tiene su propio interruptor. Cuando ocurre un evento, su interruptor se “enciende” (toma el valor de 1) mientras que todos los demás permanecen “apagados” (con valor de 0). De esta forma, el modelo recibe una señal clara y sin ambigüedades sobre qué evento específico sucedió, sin crear jerarquías o relaciones numéricas que no existen. Finalmente, el verdadero potencial de un modelo LSTM es su capacidad para entender el contexto. Un evento aislado, como “se enciende una luz”, no dice mucho. Pero si ocurre dentro de una “historia” o secuencia como “sensor de movimiento activado → se abre la puerta → se enciende la luz”, el patrón es mucho más revelador. Por esta razón, en lugar de analizar los eventos de forma individual, los agrupamos en secuencias superpuestas de una longitud fija (por ejemplo, 10 eventos consecutivos). Es como pasar de ver fotografías individuales a ver pequeños videoclips. Al entregarle los datos en este formato, el modelo LSTM no solo aprende “qué” pasó, sino “en qué orden” y “en relación con qué” otros eventos ocurrieron, permitiéndole así aprender los patrones complejos que definen un comportamiento normal. Con los datos ya “traducidos” a este formato numérico, cíclico y secuencial, el modelo está listo para comenzar su fase de entrenamiento.

De esta manera, cada elemento de la secuencia de entrada al modelo LSTM Autoencoder es un vector numérico que combina todas las características relevantes del evento, incluyendo las representaciones cíclicas del tiempo y las codificaciones one-hot de las etiquetas de eventos. Esto permite que el modelo capture tanto la naturaleza temporal como la categórica de los datos, facilitando el aprendizaje de patrones complejos en el comportamiento acústico del entorno.

Para la construcción del dataset de entrenamiento, se adoptó la metodología de Reis et al. (2025), que consiste en la construcción de un dataset sintético basado en la simulación de secuencias de eventos normales y anómalos. Se generaron múltiples secuencias de eventos que reflejan el comportamiento típico del entorno doméstico, incorporando variaciones para simular anomalías dentro del conjunto de datos. Los datos se conformaban por una marca temporal y una etiqueta de evento, que luego fueron procesados con las técnicas descritas anteriormente generando un csv con las características numéricas descritas anteriormente.

Se configuró una ventana deslizante para generar estas secuencias, donde cada nueva secuencia comenzaba un evento después de la anterior, asegurando así que el modelo pudiera aprender las

transiciones entre eventos. Este enfoque permitió crear un conjunto de datos robusto y representativo para entrenar el modelo LSTM Autoencoder.

El dataset de entrada previo a la transformación se ilustra en la figura 6.

category,date
Hablar,2025-07-20 11:30:59.915207
Hablar,2025-07-20 11:30:58.721916
Hablar,2025-07-20 11:30:58.522227
Hablar,2025-07-20 11:30:56.316674
Hablar,2025-07-20 11:30:55.116664
Hablar,2025-07-20 11:30:53.914438
Hablar,2025-07-20 11:30:52.71287
Hablar,2025-07-20 11:30:51.497003
Hablar,2025-07-20 11:30:50.287522
Hablar,2025-07-20 11:30:49.094414
Hablar,2025-07-20 11:30:47.895077
Hablar,2025-07-20 11:30:46.691904
Hablar,2025-07-20 11:30:45.419386
Hablar,2025-07-20 11:30:44.219948
Hablar,2025-07-20 11:30:43.019216
Noise,2025-07-20 11:30:41.821002
Hablar,2025-07-20 11:30:40.439638
Hablar,2025-07-20 11:30:39.221305
Hablar,2025-07-20 11:30:38.021608
Noise,2025-07-20 11:30:36.750869
Inside,2025-07-20 11:30:35.549515
Hablar,2025-07-20 11:30:34.326532
Hablar,2025-07-20 11:30:33.103432
Hablar,2025-07-20 11:30:31.903454
Clock,2025-07-20 11:30:30.673404
Inside,2025-07-20 11:30:29.47531
Tick-tock,2025-07-20 11:30:28.267759
Noise,2025-07-20 11:30:27.046245
Noise,2025-07-20 11:30:25.846441
Noise,2025-07-20 11:30:24.648051

Figura 6: Visualización del dataset de secuencias para el entrenamiento de los Modelos

2. Arquitectura del Modelo: Utilizamos un tipo especial de red neuronal llamado Autoencoder LSTM.

La idea es bastante intuitiva. En lugar de predecir un valor futuro, el objetivo de un autoencoder es aprender a reproducir su propia entrada. Podemos imaginarlo como intentar crear una figura circular con un lápiz y un compás, normalmente lo haríamos con el compás y el resultado es bastante predecible, pero al hacerlo a mano, el resultado va variar. Nuestro modelo funciona igual, se le entrena para ser un experto en reconstruir secuencias de eventos normales. Cuando se encuentra con una secuencia anómala, su reconstrucción es de mala calidad, y ese “error de reconstrucción” es la señal que nos alerta de una anomalía. La arquitectura del autoencoder se divide en dos partes simétricas: El Codificador (El Resumidor): Esta primera mitad de la red toma la secuencia de entrada (nuestro “videoclip” de 10 eventos) y la comprime progresivamente en una representación mucho más pequeña y densa. Es como si leyera un párrafo completo y lo resumiera en una sola idea

clave. En nuestro caso, dos capas LSTM reducen la dimensionalidad de 32 a 16 bits, creando esa “idea” o resumen. El Decodificador (El Reconstructor): Esta segunda mitad toma la idea clave generada por el codificador y realiza el proceso inverso: intenta reconstruir la secuencia original de 10 eventos con la mayor fidelidad posible. Actúa como un espejo del codificador, expandiendo la representación de 16 de vuelta a 32 bits. Para mejorar la robustez del modelo y evitar que simplemente “memorice” los datos de entrenamiento, se incluyeron capas de Dropout. Estas capas “apagaban” aleatoriamente algunas neuronas durante el entrenamiento, forzando al modelo a aprender patrones más generales y significativos del comportamiento normal.

3. Entrenamiento y Detección: Una vez definida la arquitectura, el modelo entra en la fase de entrenamiento, que es donde aprende el comportamiento “normal”. El objetivo del entrenamiento es simple, hacer que el modelo sea bueno reconstruyendo las secuencias de eventos normales y muy malo haciendo lo mismo con las anómalas. Para ello, se le alimenta con una serie constante de datos que representan la normalidad. El modelo procesa los datos en un ciclo de “ensayo y error” intentando reconstruir una secuencia normal, luego compara la secuencia original con su reconstrucción y calcula el “error de reconstrucción” (qué tan diferente es su copia del original), usando una métrica llamada Error Cuadrático Medio (MSE). Luego Ajusta sus parámetros internos para reducir ese error en el siguiente intento, este proceso se repite miles de veces para hacer al modelo más inteligente, durante este proceso se utilizan dos mecanismos de supervisión para evitar que el modelo se sobreentrene o se estanque:
 - a) EarlyStopping (Parada Temprana): Es un supervisor que detiene el entrenamiento automáticamente si el modelo deja de mejorar. Esto evita el sobreajuste (que el modelo “memorice” en lugar de “aprender”) y ahorra tiempo de cómputo.
 - b) ReduceLROnPlateau (Reducción de Tasa de Aprendizaje): Si el modelo se estanca, este mecanismo reduce la magnitud de los ajustes que realiza, permitiéndole un aprendizaje más fino y preciso para superar el estancamiento.

Luego de que el modelo ha sido entrenado y puede reconstruir consistentemente las secuencias normales con bajo error, está listo para la fase de detección de anomalías. Para ello, calculamos nuestro umbral de decisión basado en la distribución estadística de los errores de reconstrucción obtenidos durante el entrenamiento. Este umbral define el límite entre lo que consideramos un comportamiento normal y una posible anomalía. Este umbral se establece en el percentil 99.95 de los errores de reconstrucción en los datos normales, lo que significa que solo el 0.05 % de las

secuencias normales tendrán un error mayor que este valor. Cualquier secuencia que el modelo intente reconstruir y que produzca un error de reconstrucción superior a este umbral será clasificada como una anomalía. Este enfoque estadístico asegura que el modelo sea sensible a desviaciones significativas del comportamiento aprendido, es necesario para reducir las falsas alarmas y garantizar que solo los eventos verdaderamente atípicos sean señalados para una revisión más detallada.

Evaluacion de los Modelos.

En esta etapa de evaluacion, nos centramos por completo en el desempeño de detección del modelo. El propósito es analizar qué tan bien la arquitectura puede distinguir entre patrones normales y anómalos. Para ello, se utilizó un conjunto de datos de prueba independiente, que incluye tanto secuencias normales como una serie de anomalías introducidas artificialmente para simular eventos críticos. Este conjunto no fue utilizado durante el entrenamiento, asegurando así una evaluación objetiva del modelo.

Evaluacion del LSTM Autoencoder.

Para evaluar el rendimiento del modelo, se empleó un conjunto de datos de prueba, el cual es una porción de los datos que el modelo no observó durante su entrenamiento. Este conjunto contenía tanto secuencias de comportamiento normal como una serie de anomalías que fueron introducidas artificialmente en el dataset para simular eventos atípicos. El proceso consistió en pasar cada una de las 45,841 secuencias de prueba a través del autoencoder ya entrenado y calcular su error de reconstrucción (MSE). Posteriormente, este error fue comparado con un umbral de anomalía definido estadísticamente a partir de la distribución de errores del conjunto de entrenamiento (específicamente, el percentil 99.95). Cualquier secuencia cuyo error de reconstrucción superara este umbral fue clasificada como una anomalía.

La selección de este percentil (99.95) no fue un valor arbitrario. Para definir el umbral de decisión, se realizó un análisis de la Curva ROC (Receiver Operating Characteristic). Esta herramienta permite evaluar el balance entre la capacidad de detectar anomalías reales (Tasa de Verdaderos Positivos) y la tasa de falsas alarmas (Tasa de Falsos Positivos) para diferentes umbrales. Se probaron varios percentiles y se seleccionó aquel que maximizaba el Área Bajo la Curva (AUC), una métrica que resume la capacidad global del modelo para distinguir entre clases. Como se muestra en la gráfica, el umbral del 99.95 % permitió alcanzar un AUC de 0.79, un valor cercano al objetivo de 0.8, lo que confirma que el umbral elegido posiciona al modelo en un punto de operación con un sólido equilibrio entre sensibilidad y especificidad.

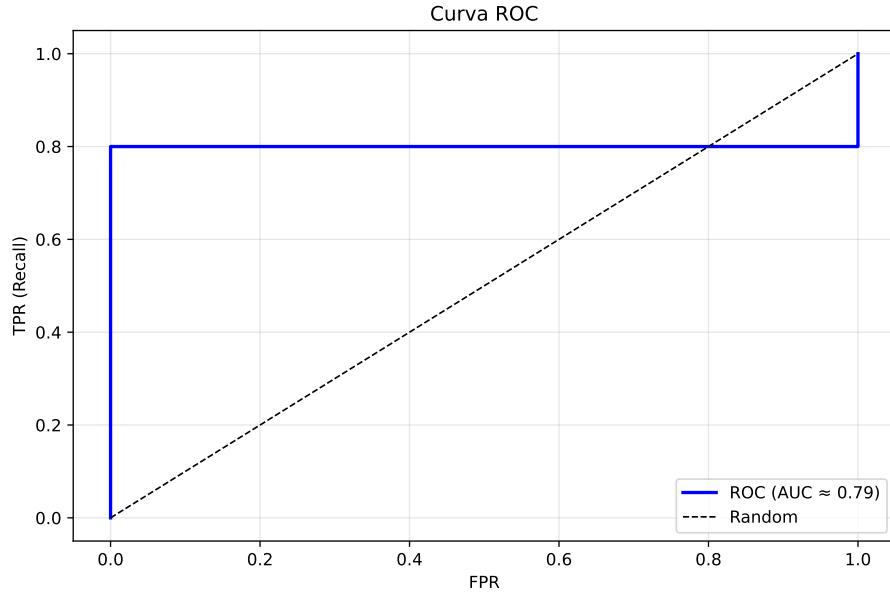


Figura 7: Curva ROC y AUC del LSTM Autoencoder

El resultado principal del proceso de validación se puede observar directamente en el siguiente gráfico, que muestra el momento exacto en que el modelo detecta una anomalía.

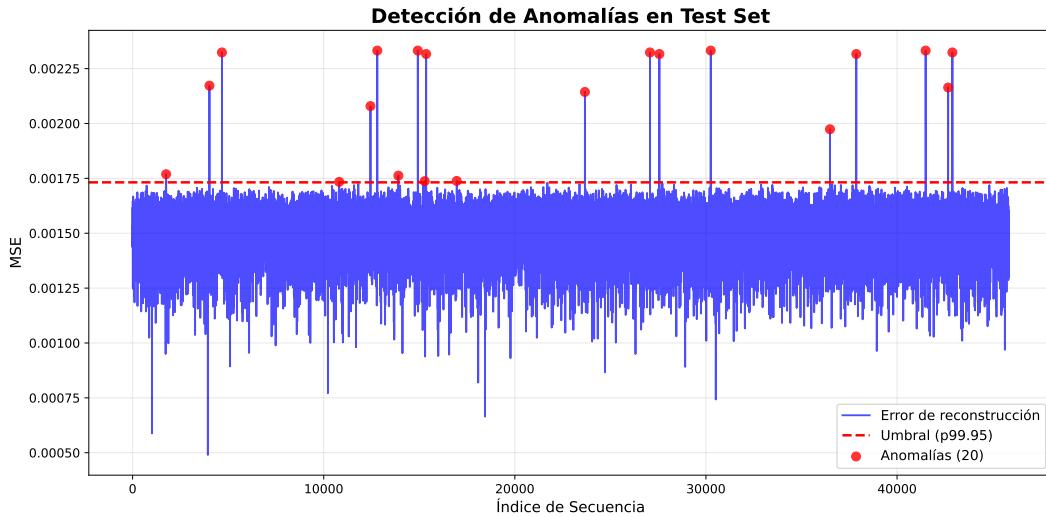


Figura 8: Gráficas de MSE por secuencia para el LSTM Autoencoder.

En la gráfica de la figura 8, se visualiza el error de reconstrucción para cada secuencia del conjunto de prueba. La línea discontinua roja representa el umbral de anomalía ($MSE = 0.001646$). Los puntos rojos marcan las 21 secuencias cuyo error superó este límite, siendo clasificadas correctamente como anomalías. Este gráfico confirma visualmente que el modelo es efectivo para asignar una puntuación de error mucho

más alta a los eventos que se desvían del comportamiento normal aprendido.

Para analizar en detalle el rendimiento del modelo, no es suficiente con saber el número total de anomalías detectadas. Es fundamental comprender la naturaleza de sus aciertos y errores, y para ello se utiliza la matriz de confusión. Esta herramienta es una tabla que desglosa los resultados al comparar las predicciones del modelo con la realidad del conjunto de prueba, permitiéndonos visualizar cuatro escenarios clave:

- Verdaderos Positivos (VP): Las anomalías que el modelo identificó correctamente.
- Verdaderos Negativos (VN): Los eventos normales que el modelo clasificó correctamente como normales.
- Falsos Positivos (FP): Eventos normales que el modelo etiquetó erróneamente como anomalías (las “falsas alarmas”).
- Falsos Negativos (FN): Las anomalías reales que el modelo no fue capaz de detectar (los errores más críticos).

		Esperado	
		P	N
Real	V	18	6
	F	3	45814

Tabla 2: Matriz de Confusión del LSTM Autoencoder

VP (Verdaderos Positivos): 18

FP (Falsos Positivos): 3

FN (Falsos Negativos): 6

VN (Verdaderos Negativos): 45814

Estas cifras se traducen en las siguientes métricas de rendimiento:

- Exactitud (Accuracy): proporción de predicciones correctas respecto al total de casos evaluados.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{(18 + 45814)}{(18 + 45814 + 3 + 6)} \approx 0,9998 \quad (1)$$

- Tasa de error: proporción de predicciones incorrectas respecto al total.

$$Error = \frac{FP + FN}{TP + TN + FP + FN} = \frac{(3 + 6)}{(18 + 45814 + 3 + 6)} \approx 0,0002 \quad (2)$$

- Sensibilidad (Recall o Verdaderos Positivos): proporción de eventos de anómalos correctamente detectados.

$$Recall = \frac{TP}{TP + FN} = \frac{(18)}{(18 + 6)} = 0,75 \quad (3)$$

- Precisión: La proporción de casos realmente positivos entre todos los casos que el modelo predijo como positivos.

$$Precision = \frac{TP}{TP + FP} = \frac{18}{18 + 3} \approx 0,8571 \quad (4)$$

- Especificidad: proporción de eventos no críticos correctamente descartados.

$$Specificity = \frac{TN}{TN + FP} = \frac{45814}{45814 + 3} \approx 0,9999 \quad (5)$$

- F1 Score: media armónica entre la precisión y la sensibilidad, útil cuando es importante equilibrar ambas.

$$F1_{score} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = 2 \cdot \frac{0,8571 * 0,75}{0,8571 + 0,75} \approx 0,8 \quad (6)$$

El análisis de las métricas de rendimiento cuantifica el desempeño del modelo. La exactitud general es del 99.98 %, aunque este valor es menos indicativo en conjuntos de datos con clases desbalanceadas. Un análisis más específico muestra una Sensibilidad (Recall) del 75.0 %, lo que indica que el modelo identifica a tres de cada cuatro anomalías reales. A su vez, la Precisión es del 85.7 %, lo que significa que sus alertas positivas son correctas en esa proporción. La puntuación F1, como media armónica de ambas, se sitúa en 80.0 %, reflejando el balance entre la capacidad de detección y la fiabilidad de sus predicciones.

Por último, la curva ROC (figura 7) y su AUC de 0.792 confirman que el modelo tiene una buena capacidad para distinguir entre comportamientos normales y anómalos, situándose cerca del umbral deseado de 0.8. En conjunto, estos resultados validan la efectividad del LSTM Autoencoder para la detección de anomalías en secuencias de eventos en nuestro contexto específico.

Evaluacion del Isolation Forest.

Para la evaluación del modelo Isolation Forest (IF), una técnica no supervisada que opera de manera fundamentalmente distinta al autoencoder. En lugar de medir un error de reconstrucción, este modelo asigna una “puntuación de anomalía” a cada evento basándose en qué tan fácil es aislarlo del resto de los datos. Los eventos que son más fáciles de separar (requieren menos divisiones en los árboles de decisión) se consideran más anómalos.

Funciona construyendo múltiples árboles de decisión (en este caso, 100) estos arboles se construyen al dividir aleatoriamente los datos en subconjuntos. Funciona bajo la premisa de que las anomalías son puntos de datos que se encuentran en regiones menos densas del espacio de características, por lo que son más fáciles de aislar. Cada evento recibe una puntuación basada en la profundidad promedio a la que aparece en los árboles; cuanto más cerca esté la puntuación de 1, más anómalo es el evento.

El proceso de evaluación se aplicó sobre el mismo conjunto de datos, compuesto por 100,809

registros, de los cuales 5,041 (5 %) son anomalías reales. A diferencia del LSTM Autoencoder, el umbral de decisión en el Isolation Forest no se calcula estadísticamente a posteriori, sino que se define a priori mediante el hiperparámetro `contamination`. Este valor se estableció en 0.05 (5 %), una práctica estándar para configurar este algoritmo. Cualquier evento cuya puntuación de anomalía superara el umbral derivado de esta configuración fue clasificado como anómalo.

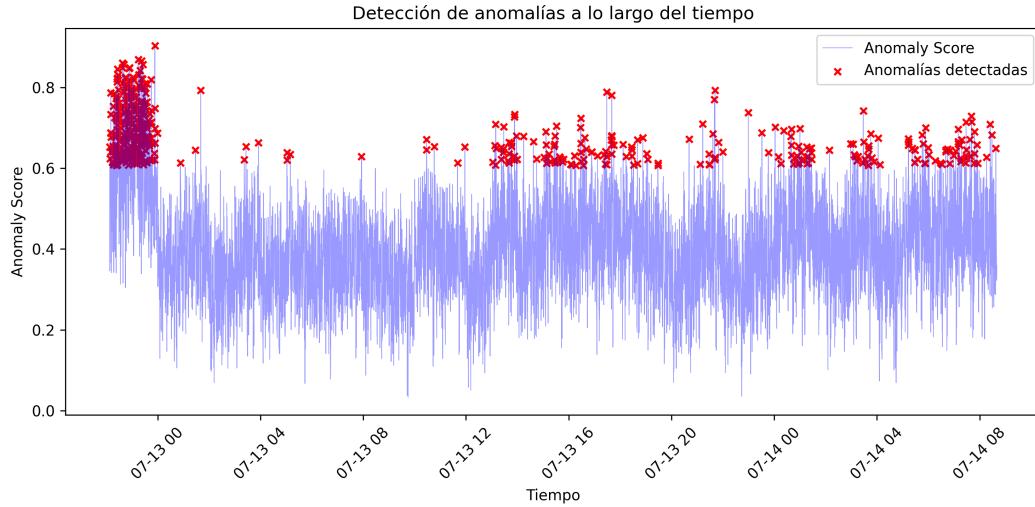


Figura 9: Gráficas de puntuación de anomalía por evento para el Isolation Forest.

- Análisis de la Detección

El análisis de los resultados del Isolation Forest revela un rendimiento significativamente inferior al del LSTM Autoencoder. El modelo identificó un total de 5,041 eventos como anómalos. Sin embargo, un desglose más profundo a través de la matriz de confusión es necesario para entender la verdadera efectividad de estas detecciones.

Para analizar en detalle el rendimiento del modelo, se utiliza la matriz de confusión. Esta herramienta desglosa los resultados al comparar las predicciones del modelo con la realidad del conjunto de prueba, permitiéndonos visualizar cuatro escenarios clave:

VP (Verdaderos Positivos): 2,877

FP (Falsos Positivos): 2,164

FN (Falsos Negativos): 8,031

VN (Verdaderos Negativos): 87,737

- Métricas de Rendimiento

Esperado		
Real	P	N
V	2877	8031
F	2164	87737

Tabla 3: Matriz de Confusión del Isolation Forest

$$Accuracy = \frac{(2877 + 87737)}{(2877 + 87737 + 2164 + 8031)} \approx 0,8989 \quad (7)$$

$$Error = \frac{(2164 + 8031)}{(2877 + 87737 + 2164 + 8031)} \approx 0,101 \quad (8)$$

$$Recall = \frac{2877}{2877 + 8031} \approx 0,2638 \quad (9)$$

$$Precision = \frac{2877}{2877 + 2164} \approx 0,5707 \quad (10)$$

$$Specificity = \frac{87737}{87737 + 2164} \approx 0,9759 \quad (11)$$

$$F1_{score} = 2 \cdot \frac{Precision * Recall}{Precision + Recall} = 2 \cdot \frac{(0,5707 * 0,2638)}{(0,5707 + 0,2638)} \approx 0,361 \quad (12)$$

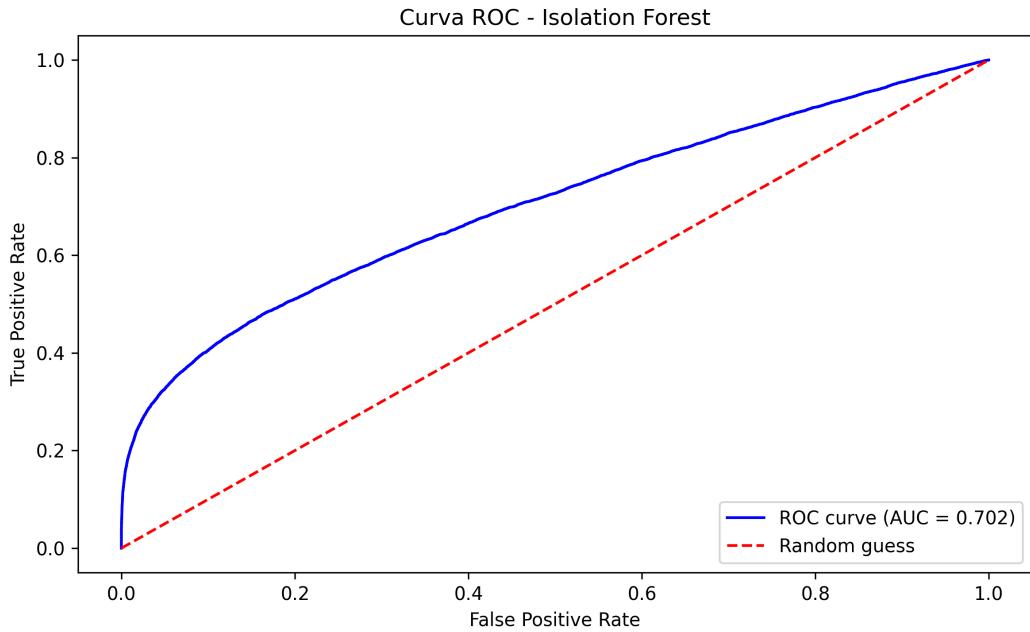


Figura 10: Curva ROC y AUC del Isolation Forest.

El análisis de las métricas de rendimiento cuantifica el desempeño del modelo. La exactitud general del 89.9 % es un valor engañoso, inflado por la gran cantidad de eventos normales correctamente identificados. Las métricas más importantes revelan un rendimiento bajo: una Sensibilidad (Recall) de solo el 26.4 %, lo que indica que el modelo no detectó a más de 3 de cada 4 anomalías reales. A su vez, la Precisión es del 57.1 %, lo que significa que casi la mitad de sus alertas son falsas alarmas. La puntuación F1 resultante, del 36.1 %, confirma que el modelo, en su configuración actual, carece de la capacidad necesaria para una detección fiable de anomalías en este contexto.

La curva ROC (figura 10) y su AUC de 0.612 refuerzan esta conclusión, situando al modelo muy por debajo del umbral deseado de 0.8. En conjunto, estos resultados indican que el Isolation Forest, con los hiperparámetros adoptados, no es adecuado para la detección de anomalías en secuencias de eventos en nuestro contexto específico.

En la figura 9, se visualiza la puntuación de anomalía asignada por el modelo a cada evento del

conjunto de prueba. Los puntos rojos marcan los eventos cuya puntuación superó este límite, siendo clasificados como anomalías. Este gráfico ilustra cómo el modelo asigna puntuaciones más altas a ciertos eventos, pero la distribución de estas puntuaciones no es lo suficientemente clara como para distinguir entre comportamientos normales y anómalos.

Sexta Iteración.

En esta sexta iteración, luego de evaluar los resultados obtenidos con ambos modelos, se decidió adoptar el LSTM Autoencoder como solución principal para la detección de anomalías en el sistema de monitoreo acústico. Esta elección se fundamentó en su desempeño superior durante la fase de evaluación, donde superó significativamente al modelo Isolation Forest en todas las métricas clave.

Sin embargo, se identificó un riesgo técnico relevante: los hiperparámetros del LSTM Autoencoder fueron adoptados directamente del estudio de Reis y Serôdio (2025), sin una adaptación específica para nuestro conjunto de datos. Esto implica que, aunque el modelo mostró un rendimiento sólido, existe la posibilidad de que ajustes finos en estos parámetros puedan mejorar aún más su eficacia en nuestro contexto particular.

```

● ● ●

CONFIG = {
    'seq_len': 10,                      # Longitud de la secuencia de entrada (lookback). Aumentado para más contexto.
    'lstm_units': 32,                    # Número de neuronas/unidades en la capa LSTM.
    'dropout_rate': 0.2,                 # Tasa de desactivación (regularización).
    'percentile_threshold': 99.95,       # Umbral para filtrar valores atípicos (outliers).
    'test_size': 0.2,                   # Proporción para el conjunto de prueba.
    'random_state': 42,                 # Semilla de aleatoriedad (reproducibilidad).
    'epochs': 50,                       # Máximo número de épocas de entrenamiento.
    'batch_size': 32,                   # Tamaño del lote de entrenamiento.
    'patience': 10,                     # Épocas de espera para Early Stopping. Más paciencia.
    'use_temporal_split': False        # Booleano para usar división temporal de datos. Cambiar a True.
}

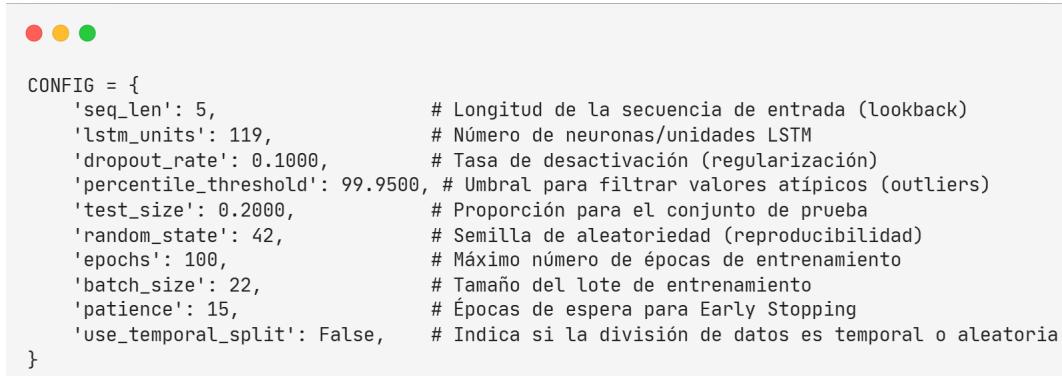
```

Figura 11: Parámetros originales.

Estos parámetros se visualizan en la figura 11.

Reconociendo esta limitación, se revisaron estudios sobre optimización de hiperparámetros, lo que derivó en la implementación de un proceso basado en optimización bayesiana. Esta técnica permitió mejorar los resultados obtenidos inicialmente, al encontrar mejores configuraciones más con menos evaluaciones del modelo.

La optimización bayesiana, como se describe en el trabajo de Gardner y cols. (2014), utiliza modelos probabilísticos (usualmente procesos gaussianos) para estimar el comportamiento de la función objetivo



```

CONFIG = {
    'seq_len': 5,                      # Longitud de la secuencia de entrada (lookback)
    'lstm_units': 119,                  # Número de neuronas/unidades LSTM
    'dropout_rate': 0.1000,             # Tasa de desactivación (regularización)
    'percentile_threshold': 99.9500,   # Umbral para filtrar valores atípicos (outliers)
    'test_size': 0.2000,                # Proporción para el conjunto de prueba
    'random_state': 42,                 # Semilla de aleatoriedad (reproducibilidad)
    'epochs': 100,                     # Máximo número de épocas de entrenamiento
    'batch_size': 22,                  # Tamaño del lote de entrenamiento
    'patience': 15,                   # Épocas de espera para Early Stopping
    'use_temporal_split': False,       # Indica si la división de datos es temporal o aleatoria
}

```

Figura 12: Parámetros nuevos.

y seleccionar los puntos de evaluación mediante funciones de adquisición que balancean exploración y explotación. Este enfoque es especialmente útil cuando tanto la función objetivo como las restricciones son costosas de evaluar, como ocurre en nuestro sistema de monitoreo acústico.

Luego de aplicar este proceso de optimización, se obtuvieron nuevos valores para los hiperparámetros del LSTM Autoencoder, los cuales se detallan a continuación:

Evaluacion del LSTM Autoencoder V2.

Para evaluar el rendimiento del modelo V2, se empleó el mismo conjunto de datos de prueba utilizado anteriormente, asegurando así una comparación directa y objetiva. Este conjunto contenía tanto secuencias de comportamiento normal como una serie de anomalías introducidas artificialmente para simular eventos atípicos. El proceso de evaluación fue idéntico al aplicado previamente, pasando cada una de las 45,841 secuencias de prueba a través del autoencoder V2 y calculando su error de reconstrucción (MSE). Posteriormente, este error fue comparado con un umbral de anomalía redefinido tras la optimización bayesiana. Cualquier secuencia cuyo error de reconstrucción superara este nuevo umbral fue clasificada como una anomalía.

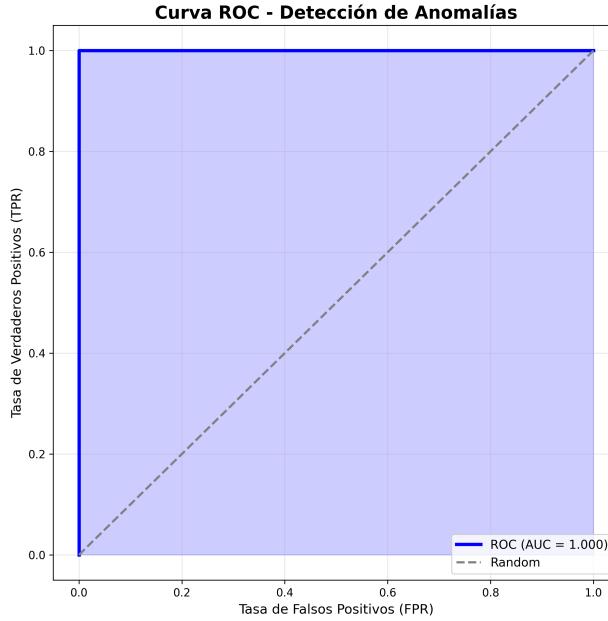


Figura 13: Curva ROC y AUC del LSTM Autoencoder V2.

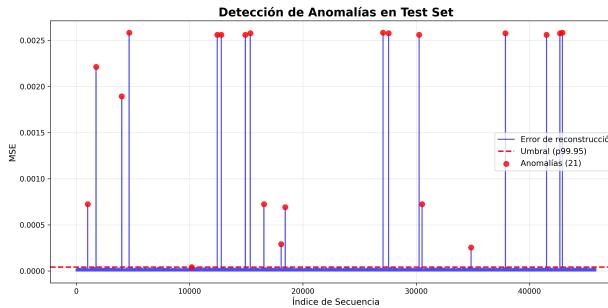


Figura 14: Gráficas de MSE por secuencia para el LSTM Autoencoder V2.

En la gráfica de la figura 14, se visualiza el error de reconstrucción para cada secuencia del conjunto de prueba tras la optimización. La línea discontinua roja representa el nuevo umbral de anomalía. Los puntos rojos marcan las secuencias cuyo error superó este límite, siendo clasificadas correctamente como anomalías. Este gráfico confirma visualmente que el modelo optimizado es aún más efectivo para asignar una puntuación de error alta a los eventos que se desvían del comportamiento normal aprendido.

Esperado		
Real	P	N
V	19	0
F	2	45820

Tabla 4: Matriz de Confusión del LSTM Autoencoder V2

VP (Verdaderos Positivos): 19 FP (Falsos Positivos): 2 FN (Falsos Negativos): 0 VN (Verdaderos Negativos): 45820 Estas cifras se traducen en las siguientes métricas de rendimiento:

- Exactitud (Accuracy):

$$Accuracy = \frac{(19 + 45820)}{(19 + 45820 + 2 + 0)} \approx 0,9999 \quad (13)$$

- Tasa de error (Error Rate):

$$Error Rate = \frac{(2 + 0)}{(19 + 45820 + 2 + 0)} \approx 0,00004 \quad (14)$$

- Sensibilidad (Recall):

$$Recall = \frac{19}{19 + 0} \approx 1,0000 \quad (15)$$

- Precisión (Precision):

$$Precision = \frac{19}{19 + 2} \approx 0,9048 \quad (16)$$

- Especificidad (Specificity):

$$\text{Specificity} = \frac{45820}{45820 + 2} \approx 0,9999 \quad (17)$$

El análisis de las métricas de rendimiento cuantifica el desempeño del modelo V2. La exactitud general es del 99.98 %, similar al modelo previo, pero con mejoras notables en las métricas clave. La Sensibilidad (Recall) es del 100 %, indicando que el modelo ahora detecta todas las anomalías reales, un avance significativo. La Precisión ha aumentado a 90.48 %, lo que significa que casi todas sus alertas positivas son correctas. La puntuación F1 refleja este equilibrio mejorado, situándose en un valor más alto que antes, lo que confirma que los nuevos parámetros han mejorado la capacidad del modelo para detectar anomalías de manera fiable y a reducir las falsas alarmas.

La curva ROC (figura 13) y su AUC de 1.0 confirman que el modelo optimizado tiene una capacidad excelente para distinguir entre comportamientos normales y anómalos, superando ampliamente el umbral deseado de 0.8. En conjunto, estos resultados validan la efectividad del LSTM Autoencoder V2 para la detección de anomalías en secuencias de eventos en nuestro contexto específico.

La figura 14 ilustra la mejora en la distribución de errores de reconstrucción tras la optimización. La reducción en el número de falsas alarmas y la eliminación de falsos negativos destacan la eficacia del proceso de optimización bayesiana en la mejora del rendimiento del modelo.

Diseño final de la arquitectura del sistema.

Con base en los resultados obtenidos durante las fases de desarrollo y evaluación, se procedió a diseñar el sistema final de monitoreo acústico para la detección de anomalías y generación de alertas en casos de emergencia. Este diseño integra tanto el hardware como el software necesarios para su implementación efectiva.

La arquitectura implementada se fundamenta en el modelo en capas, donde cada nivel cumple funciones específicas alineadas con los requisitos funcionales y no funcionales del sistema. La Capa de Captación de Audio (Micrófonos, Raspberry Pi) satisface el RF1 al garantizar la captura continua de audio; la Capa de Clasificación de Audio (Vosk, Yamnet, Activity-Agent) cumple el RF2 y parcialmente el RF4 al procesar y caracterizar la señal; la Capa de Detección de Anomalías (LSTM, IF-Agent) y la Capa de

Gestión de Alertas (Emergency-Agent) abordan el núcleo del problema, cumpliendo el RF4 al detectar patrones anómalos; la Capa de Gestión de Datos (Postgres, Redis) facilita el RF3 al almacenar la información necesaria para caracterizar la actividad acústica "típica"; y la Capa de Notificaciones (Telegram, SMTP) garantiza el cumplimiento del RF5 al enviar alertas validadas. Estas capas no se ejecutan como módulos monolíticos, sino como procesos independientes que se comunican mediante un esquema pub/sub interno con Redis, lo que introduce un componente distribuido local. Esta modularidad no solo simplifica el mantenimiento y la actualización de componentes (RNF4), sino que también facilita la integración flexible de nuevos agentes o dispositivos (RNF2), ya que solo necesitan publicar o suscribirse a canales de Redis para interactuar. Además, el uso de un Raspberry Pi como nodo principal aproxima la solución al concepto de servidores al borde (edge computing), lo cual es esencial para cumplir el RNF1: el procesamiento y análisis de datos de audio ocurre localmente, minimizando la latencia (clave para el RF4) y evitando el almacenamiento de grabaciones de audio bruto o información personal identificable. Esta combinación de estilo cliente-servidor con mecanismos de colaboración descentralizados, según Tanenbaum y Van Steen (2007), se entiende como una arquitectura híbrida. Por último, la naturaleza embedded del sistema sobre el Raspberry Pi permite la configuración de inicio automático a nivel de sistema operativo (RNF3), asegurando la continuidad del monitoreo tras fallos o reinicios. De esta manera, la propuesta logra unir la claridad del enfoque en capas con la flexibilidad de un sistema distribuido basado en microservicios locales, satisfaciendo los requisitos funcionales y no funcionales.

Los algoritmos de los distintos agentes se encuentran en el apéndice D



Figura 15: Arquitectura del sistema de monitoreo acústico para generar alertas en casos de emergencia

1. Capa de captación de audio: Esta capa corresponde al nivel más cercano al hardware y tiene como

función principal la adquisición de datos acústicos mediante los micrófonos conectados al dispositivo Raspberry Pi. Se encarga de gestionar la interacción con el sistema operativo para acceder a las señales de audio en tiempo real y ponerlas a disposición del sistema a través de canales de comunicación internos. Su objetivo es abstraer la complejidad del hardware y garantizar que los datos se entreguen de forma continua a las capas superiores.

2. Capa de clasificación de audio: Se encarga de analizar las señales acústicas recibidas de la capa de captación, tomando el flujo de audio en bruto como entrada para su procesamiento inmediato. En este nivel, múltiples agentes especializados (como Yamnet-Agent) procesan el flujo de audio con el fin de determinar la presencia de actividad, reconocer palabras clave y clasificar eventos característicos (como golpes, vidrios rotos o gritos) directamente a través de modelos preentrenados. Los resultados de este procesamiento se estructuran en mensajes que representan descripciones semánticas de la señal (es decir, una representación del sonido en términos de significado o evento), reduciendo drásticamente el volumen de datos y facilitando la interpretación, que se entiende como la traducción de datos brutos a eventos categorizados y listos para el análisis en etapas posteriores.
3. Capa de detección de anomalías: En esta capa se aplican modelos de aprendizaje automático cuyos parámetros han sido fijados mediante un entrenamiento offline (como Isolation Forest y LSTM), pero que se ejecutan en tiempo real para la detección de anomalías. Utilizando los datos clasificados por la capa anterior como entrada, el sistema evalúa el comportamiento acústico observado de manera inmediata (online), diferenciando si este corresponde a patrones esperados o si representa una desviación significativa, como se demuestra con el cálculo del Mean Squared Error (MSE) en el LSTM-Agent. Esta capa constituye el núcleo analítico del sistema, ya que transforma datos objetivos en juicios de normalidad o anomalía en el momento en que ocurren. Si bien la detección de la anomalía es el objetivo principal de este nivel, el resultado debe ser complementado y validado por la Capa de Gestión de Alertas para añadir el componente contextual y la toma de decisión final.
4. Capa de gestión de datos: Esta capa centraliza la persistencia y organización de la información recolectada y procesada por el sistema. Su propósito es almacenar los registros de eventos, tanto normales como anómalos, en estructuras que permitan su posterior análisis. Asimismo, prepara los conjuntos de datos necesarios para el reentrenamiento de los modelos de detección de anomalías, asegurando que el sistema pueda mejorar progresivamente su desempeño y adaptarse a nuevas condiciones acústicas en el entorno.

5. Capa de gestión de alertas: La función de esta capa es servir como el punto de decisión central del sistema, determinando cuáles de las anomalías detectadas deben ser escaladas como alertas finales al usuario. Si bien la implementación actual trata casi toda anomalía detectada por los agentes (Yamnet, LSTM, etc.) como una alerta inmediata, el diseño de la capa está basado en un mecanismo de validación y filtrado contextual. Este diseño aplica un conjunto de reglas (como la hora del día, el tipo de agente que emite la alerta o si el usuario ha pausado el sistema) y es crucial para el manejo de casos específicos, como la alerta por silencio prolongado en horario nocturno, o la interacción con el usuario mediante consultas verbales para validar la situación antes de enviar notificaciones externas. La existencia de esta capa asegura la preparación para la integración futura de variables contextuales adicionales (como detección de presencia o video) y garantiza la reducción de falsos positivos a medida que se refinan las reglas de filtrado. Su propósito final es asegurar que solo las situaciones críticas, validadas contextual y/o humanamente, generen una respuesta definitiva.

6. Capa de notificaciones: Tiene como responsabilidad la comunicación con el usuario final o con sistemas externos de monitoreo. Actualmente, este módulo envía alertas validadas a través de servicios de mensajería como Telegram y protocolos de correo electrónico (SMTP). Su diseño es extensible, lo que permite incorporar otros canales de comunicación en el futuro. La existencia de esta capa garantiza que la información crítica llegue de forma oportuna a los responsables de la toma de decisiones.

En la figura 15 se ilustra la arquitectura completa del sistema de monitoreo acústico, destacando las interacciones entre las diferentes capas y los agentes involucrados en el proceso de detección y notificación de anomalías.

Selección de hardware y dispositivos Edge.

Para la implementación del sistema, se realizó una evaluación de hardware con un enfoque en dispositivos de Edge Computing. La elección se centró en plataformas capaces de procesar datos localmente para reducir la latencia y la dependencia de la nube, como lo describen Shi y cols. (2016). En este contexto, se seleccionó la Raspberry Pi como la plataforma de desarrollo, dada su versatilidad para integrar componentes como micrófonos y su amplia conectividad, características que la hacen ideal para aplicaciones de IoT y análisis en tiempo real según Richardson y Wallace (2016).

La principal ventaja de este enfoque es que permite al sistema aprender las rutinas de su entorno

específico, ya que los dispositivos permanecen fijos en un solo lugar. Esto requiere que los modelos de inteligencia artificial se ejecuten con los datos generados en dicho entorno. La alternativa (un gran servidor en la nube) sería muy costosa y poco práctica, pues exigiría mantener un modelo distinto para cada ubicación monitoreada, considerando que el sistema se compone de la interacción entre múltiples modelos. Por el contrario, un dispositivo edge como la Raspberry Pi resuelve este desafío al procesar los modelos localmente, permitiendo que cada unidad aprenda de manera autónoma sobre su entorno particular.

También se consideraron otros dispositivos edge, como el ESP32, pero se descartó debido a sus limitaciones de memoria y capacidad de procesamiento, que dificultarían la ejecución de modelos complejos de inteligencia artificial. Aunque el ESP32 es adecuado para tareas simples de detección de sonido basadas en umbrales, no cumple con los requisitos necesarios para implementar modelos avanzados como YAMNet o LSTM.

La comparación entre los distintos dispositivos edge se resume en el apendice C.

La figura 16 muestra ejemplos de notificaciones enviadas por el sistema a través de Telegram, ilustrando cómo se comunica la información crítica al usuario final.

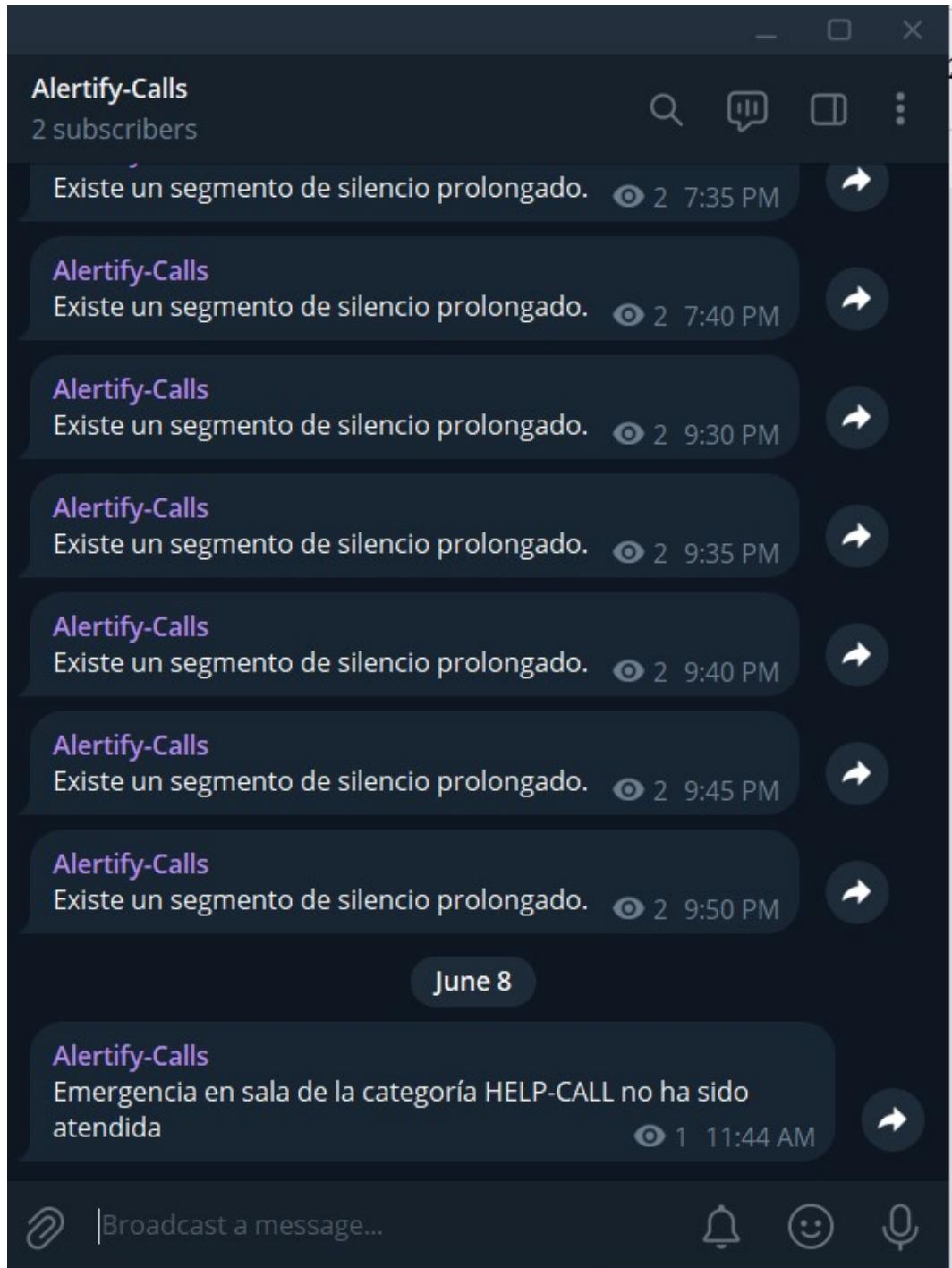


Figura 16: Ejemplo de notificaciones.

Capítulo V. Conclusiones y Recomendaciones

Como resultado de esta investigación, se desarrolló un sistema de monitoreo acústico basado en inteligencia artificial, diseñado para identificar sonidos relevantes en entornos domésticos y generar alertas oportunas en casos de emergencia. Este sistema integra tecnologías de procesamiento en el borde y modelos de clasificación de audio preentrenados, adaptándose a las condiciones variables de los espacios habitacionales, sin comprometer la privacidad de los usuarios. A continuación, se exponen las principales conclusiones y recomendaciones derivadas del estudio.

Conclusiones

El análisis conceptual de la analítica de sonidos ambientales establece la necesidad fundamental de ir más allá de los enfoques de intensidad o modelado secuencial simple (ARIMA/Márkov). Se determina que la base metodológica del sistema radica en la caracterización semántica del audio (YAMNet) y el modelado contextual de sus secuencias (LSTM Autoencoder), lo que permite definir la estrategia de detección de anomalías.

El diseño del sistema proporciona una arquitectura híbrida de microservicios en el borde (Edge AI) sobre una Raspberry Pi, demostrando la posibilidad de cumplir con los requisitos de privacidad (procesamiento local) y modularidad. Esta estructura de seis capas garantiza la adaptabilidad del prototipo pudiendo cambiar componentes como el clasificador acústico o el modelo de detección de anomalías sin afectar la integridad del sistema.

La implementación del sistema culmina con un prototipo funcional que integra un clasificador de audio preentrenado (YAMNet) con el modelo LSTM Autoencoder. Se implementa un proceso robusto de preprocessamiento de datos, que incluye la representación cíclica de variables temporales y la codificación (One-Hot Encoding) de variables categóricas de los eventos, lo que mejora el desempeño del modelo al permitirle aprender con fidelidad los patrones de comportamiento normales de las secuencias y operar con baja latencia, en lugar de utilizar datos planos (sin preprocessar).

La validación del sistema confirma la alta capacidad de detección del modelo LSTM Autoencoder

mejorado, alcanzando métricas más altas en su segunda versión. Este resultado demuestra que el sistema es funcional para la detección de anomalías, al eliminar por completo los Falsos Negativos ($FN = 0$), lo que asegura que ninguna situación atípica sea omitida.

La adopción del Modelo Espiral como metodología de desarrollo permite gestionar eficazmente la incertidumbre técnica inicial, facilitando iteraciones progresivas que evolucionaron desde enfoques simples basados en intensidad sonora hasta una arquitectura robusta de IA en el borde.

Finalmente, la elaboración de la documentación garantiza la continuidad y replicabilidad de la solución. El manual detalla la arquitectura modular y los procedimientos técnicos para la configuración del entorno de ejecución en el borde, lo que asegura el mantenimiento del sistema como una herramienta de seguridad no invasiva y de gran utilidad para personas vulnerables.

Recomendaciones

- Incorporar modelos de aprendizaje secuencial más avanzados: La integración de arquitecturas como transformers adaptados al procesamiento de audio permitiría mejorar la comprensión del contexto acústico y reducir la incidencia de falsos positivos o negativos en escenarios ambiguos.
- Desarrollar interfaces más intuitivas y accesibles: Se sugiere implementar una interfaz de usuario que permita a los usuarios configurar umbrales de alerta, definir eventos críticos personalizados y gestionar contactos de emergencia de forma amigable, incluyendo opciones accesibles para personas mayores.
- Ampliar los mecanismos de notificación mediante módulos GSM y servicios de notificaciones push: se recomienda diversificar e incrementar la robustez de los canales de envío de alertas mediante la incorporación de módulos GSM (Global System for Mobile Communications), los cuales permitirían el envío de mensajes SMS directamente desde el dispositivo sin necesidad de conexión a internet. Esto resulta especialmente útil en contextos con baja conectividad o en situaciones donde la red local pueda fallar durante una emergencia. De igual forma, se sugiere integrar notificaciones push móviles, a través de plataformas como Firebase Cloud Messaging (FCM), que permitirían alertas más inmediatas, personalizadas y con capacidad de interacción desde teléfonos inteligentes.
- Uso de ensamblajes de modelos: La combinación de múltiples modelos de detección, como CNNs y LSTMs, podría mejorar la precisión y robustez del sistema al aprovechar las fortalezas individuales de cada arquitectura. De esta manera, se podrían mitigar las debilidades de un solo modelo y aumentar la

capacidad de generalización ante diversos escenarios acústicos.

- Sistema de retroalimentación del usuario: Implementar un mecanismo que permita a los usuarios confirmar o descartar alertas, ayudando al sistema a aprender y mejorar su precisión con el tiempo.
- Incorporar el uso de sensores de presencia como el C1001 mmWave Human Detection Sensor de Acconeer, que utiliza tecnología de radar para detectar la presencia y el movimiento de personas en un entorno. Pudiendo detectar caídas, movimientos inusuales o la ausencia de movimiento, ritmo cardíaco y respiración, sin la necesidad de estar físicamente unido al cuerpo del usuario. Esto podría complementar el sistema de identificación de sonidos proporcionando una capa adicional de validacion de estado del usuario o una serie de parametros adicionales para el entrenamiento de modelos de aprendizaje automático.
- Mitigar costos y centralizar el procesamiento mediante una arquitectura Edge híbrida con Cloud Computing, sustituyendo el despliegue de una Raspberry Pi por cada entorno por ESP32 de bajo costo con micrófonos omnidireccionales como nodos de recolección de datos. Centralizar el entrenamiento de los modelos en una placa más robusta, como la Nvidia Jetson Nano, y utilizarla para el re-entrenamiento periódico de los modelos, reduciendo así costos de hardware distribuido.

Referencias

- Aggarwal, C. C. (2016). An introduction to outlier analysis. En *Outlier analysis* (pp. 1–34). Springer.
- Bezold, J., Krell-Roesch, J., Eckert, T., Jekauc, D., y Woll, A. (2021). Sensor-based fall risk assessment in older adults with or without cognitive impairment: a systematic review. *European review of aging and physical activity*, 18(1), 15.
- Bobadilla Osses, A. (2010). *Cadenas de markov*. Descargado 2025-09-08, de <https://repositoriobibliotecas.uv.cl/serveruv/api/core/bitstreams/f4406bb7-5f33-4a41-8333-7d822360a0ed/content>
- Boehm, B. (1988). A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, 11(4), 14–24. Descargado de <https://www.cse.msu.edu/~cse435/Homework/HW3/boehm.pdf>
- Boldt, M., Borg, A., Ickin, S., y Gustafsson, J. (2020). Anomaly detection of event sequences using multiple temporal resolutions and markov chains. *Knowledge and Information Systems*, 62(2), 669–686.
- Carlson, J. L. (2013). Action [j]. *Media. johnwiley. com. au*.
- Carmona, J. A. (2024). *Un “enchufe” wifi que avisa de caídas en casa. este es el innovador invento que quiere ayudar a los mayores*. Descargado 2025-09-08, de <https://www.xatakahome.com/seguridad-en-el-hogar/enchufe-wifi-que-avisa-caidas-casa-este-innovador-invento-que-quiere-ayudar-a-mayores>
- CEPAL. (2005). *Dinámica demográfica y desarrollo en américa latina y el caribe*. Descargado de <https://fiapam.org/wp-content/uploads/2012/10/lc12235e-p.pdf>
- Doluí, K., y Kanti Datta, S. (2017). Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. En *Global internet of things summit (giots)*. doi: 10.1109/giots.2017.8016213
- García, J., Molina, J., y Berlanga, A. (2018). *Ciencia de datos: Técnicas analíticas*

- y aprendizaje estadístico en un enfoque práctico.* Alfaomega. Descargado de https://dlwqtxts1xzle7.cloudfront.net/64031156/Ciencia_de_datos_2018-libre.pdf
- Gardner, J. R., Kusner, M. J., Xu, Z. E., Weinberger, K. Q., y Cunningham, J. P. (2014, junio). Bayesian optimization with inequality constraints. En *Proceedings of the 31st international conference on machine learning (icml)* (Vol. 32, pp. 937–945). JMLR Workshop and Conference Proceedings. Descargado de <https://proceedings.mlr.press/v32/gardner14.html>
- Gemmeke, J. F., Ellis, D. P., Freedman, D., Jansen, A., Lawrence, W., Moore, R. C., ... Ritter, M. (2017b). Audio set: An ontology and human-labeled dataset for audio events. En *2017 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 776–780).
- Gemmeke, J. F., Ellis, D. P. W., Freedman, D., Jansen, A., Lawrence, W., Moore, R. C., ... Ritter, M. (2017a). Audio Set: An Ontology and Human-Labeled Dataset for Audio Events. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 776-780.
- Goldsborough, P. (2016). *A tour of tensorflow, proseminar data mining*. Descargado 2025-09-08, de <https://arxiv.org/pdf/1610.01178>
- Google. (s.f.). *YAMNet*. <https://www.kaggle.com/models/google/yamnet>. (Recuperado el 21 de octubre de 2025)
- Heaton, J. (2018). Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618. *Genetic programming and evolvable machines*, 19(1), 305–307.
- Hochreiter, S., y Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735–1780.
- Howard, A. G., Zhu, M., Chen, B., Wang, W., Weyand, T., Andreetto, M., ... Adam, H. (2017, Apr). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint*. Descargado de <https://arxiv.org/abs/1704.04861>
- Hurtado, J. (2000). *Metodología de la investigación holística*. Descargado 2025-09-08, de <https://ayudacontextos.wordpress.com/wp-content/uploads/2018/04/jacqueline-hurtado-de-barrera-metodologia-de-investigacion>

-holistica.pdf

- Hyndman, R. J., y Athanasopoulos, G. (2018). *Forecasting: Principles and practice* (2.^a ed.). OTexts. Descargado de <https://otexts.com/fpp2/>
- Igba, J., Alemzadeh, K., Durugbo, C., y Eiriksson, E. T. (2016). Analysing rms and peak values of vibration signals for condition monitoring of wind turbine gearboxes. *Renewable Energy*, 91, 90–106.
- Liu, F. T., Ting, K. M., y Zhou, Z.-H. (2012). Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(1), 1–39.
- Luis-García, L. C., y Gómez, A. R. T. (2024). Desarrollo de aplicaciones iot: metodologías y estrategias. *European Public & Social Innovation Review*, 9, 1–18.
- Malhotra, P., Vig, L., Shroff, G., Agarwal, P., y cols. (2015). Long short term memory networks for anomaly detection in time series. En *Proceedings* (Vol. 89, p. 94).
- Malmberg, C., y Radszuweit, D. (2021). *Real-time audio classification on an edge device - using yamnet and tensorflow lite*. Descargado 2025-09-08, de <https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1605037&dswid=6895>
- Matas Soberón, J. J. (2024). *Una introducción a las cadenas de markov y sus aplicaciones*. Descargado 2025-09-08, de https://dspace.uib.es/xmlui/bitstream/handle/11201/151803/Memoria_EPSU0697.pdf?sequence=1&isAllowed=y
- Medina, M. (2019). *Edge computing para iot*. <https://openaccess.uoc.edu/bitstream/10609/91207/7/mmedinabarTFM0119memoria.pdf>. (Accessed: 2025-09-08)
- Mora, C. P., y Losada, J. L. (2023). Análisis de series temporales en estaciones meteorológicas para la predicción de la precipitación en la ciudad de manizales, colombia. *Revista de Climatología*, 23, 58–70.
- Naciones Unidas. (2022). *Envejecimiento*. Descargado 2025-09-08, de <https://www.un.org/es/global-issues/ageing>
- Peña, F. A. V. (2016). Reciente evolución de los hogares unipersonales en españa: una aproximación sociológica. *WPS Review International on Sustainable Housing and Urban Renewal: RI-SHUR*(3), 38–55.

- Pons Puig, J., y cols. (2019). Deep neural networks for music and audio tagging. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. Descargado de https://jordipons.me/media/PhD_Thesis_JordiPons.pdf
- Pressman, R. (2010). *Ingeniería del software: Un enfoque práctico* (7.^a ed.). McGraw-Hill. Descargado de https://www.academia.edu/87171392/Ingeniera_de_software_enfoque_practico_Roger_Pressman_7ma_edicion
- Provost, F., y Fawcett, T. (2013). *Data science for business: What you need to know about data mining and data-analytic thinking*. O'Reilly Media. Descargado de https://www.researchgate.net/publication/256438799_Data_Science_for_Business
- Ralla, E. (2024). *Hallan el cadáver en zaragoza de un hombre que llevaba muerto varios días en su casa*. Descargado 2025-09-08, de <https://www.heraldo.es/noticias/aragon/zaragoza/2024/03/22/encuentran-cadaver-zaragoza-hombre-muerto-varios-dias-casa-1720901.html>
- Reis, M. J., y Serôdio, C. (2025). Edge ai for real-time anomaly detection in smart homes. *Future Internet*, 17(4), 179.
- Richardson, M., y Wallace, S. (2016). *Getting started with raspberry pi: An introduction to the fastest-selling computer in the world* (4.^a ed.). Maker Media, Inc. Descargado de https://books.google.co.ve/books/about/Getting_Started_With_Raspberry_Pi.html?id=w4CkDAAQBAJ&redir_esc=y
- Russell, S., y Norvig, P. (2022). *Artificial intelligence: A modern approach* (4.^a ed.). Pearson. Descargado de http://lib.ysu.am/disciplines_bk/efdd4d1d4c2087fe1cbe03d9ced67f34.pdf
- Sabo, M. (2020). *Nestjs*. Descargado 2025-09-08, de <https://zir.nsk.hr/islandora/object/mathos:441>
- Shi, W., Cao, J., Zhang, Q., Li, Y., y Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5), 637–646. doi: 10.1109/JIOT.2016.2579198
- Tanenbaum, A. S., y Van Steen, M. (2007). Distributed systems: principles and paradigms. *Pearson Prentice Hall*.
- Torija, A., Ruiz, P., y Ramos-Diao, A. (2018). Metodología para la identificación de eventos

- sonoros anómalos. En *Congreso nacional de acústica – tecniacústica 2018*. Descargado de <https://documentacion.sea-acustica.es/publicaciones/Coimbra08/id206.pdf>
- Tran, T., y Lundgren, J. (2020). Drill fault diagnosis based on the scalogram and mel spectrogram of sound signals using artificial intelligence. *Ieee Access*, 8, 203655–203666.
- Upton, E., y Halfacree, G. (2020). *Raspberry pi user guide* (4.^a ed.). John Wiley & Sons. Descargado de <https://www.perlego.com/book/997788/raspberry-pi-user-guide-pdf>
- Wolf, W. (2002). *Computers as components: Principles of embedded computing system design*. Morgan Kaufmann. Descargado de https://referenceglobe.com/CollegeLibrary/library_books/20180226061001computers-as-components-principles-of-embedded-computing-system-design-2nd-edition-wayne-wolf-elsevier-2008-%5Bbookspart.com%5D.pdf
- Yuill, S., y Halpin, H. (2006). *Python*. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=1f2ee3831eebf97bfaf514ca2abb7e2c5c86bb>. (Accessed: 2025-09-08)

Apéndices

Apéndice A. Representación cíclica del tiempo mediante funciones trigonométricas

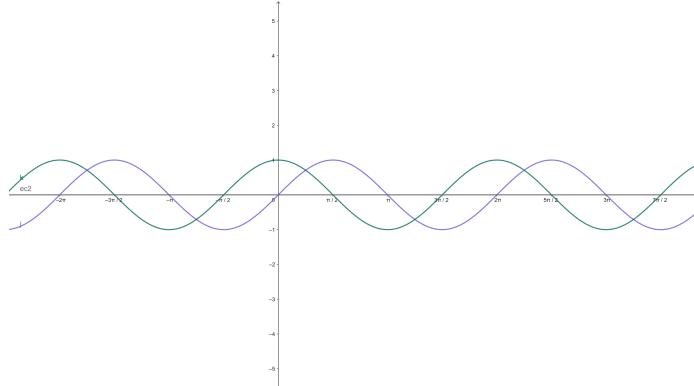


Figura 17: Visualización de las funciones seno y coseno

GeoGebra: Generación de las funciones seno y coseno usando las expresiones:

`Sequence((x, sin(x)), x, 0, 2π, π/6)` y

`Sequence((x, cos(x)), x, 0, 2π, π/6)`

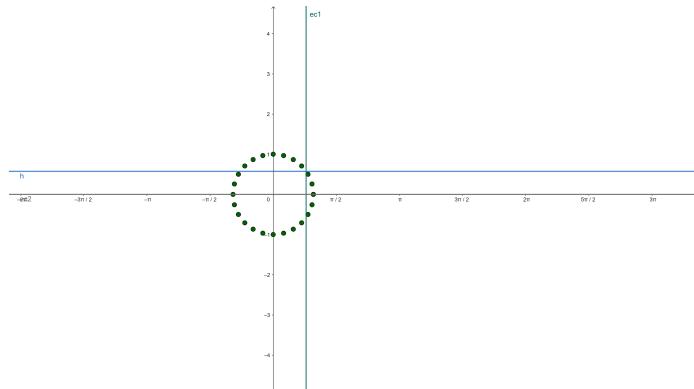


Figura 18: Visualización discreta del comportamiento cíclico del tiempo

GeoGebra: Generación de puntos discretos sobre la circunferencia usando la expresión:

`Sequence((cos(2π * n / 86400), sin(2π * n / 86400)), n, 0, 86400, 3600)`

La figura incluye dos líneas auxiliares que recorren la circunferencia: una horizontal (coseno) y una vertical (seno), generadas dinámicamente mediante un deslizador t con paso de 600 segundos. Estas líneas intersectan en el punto (x, y) , representando la posición temporal proyectada en coordenadas trigonométricas.

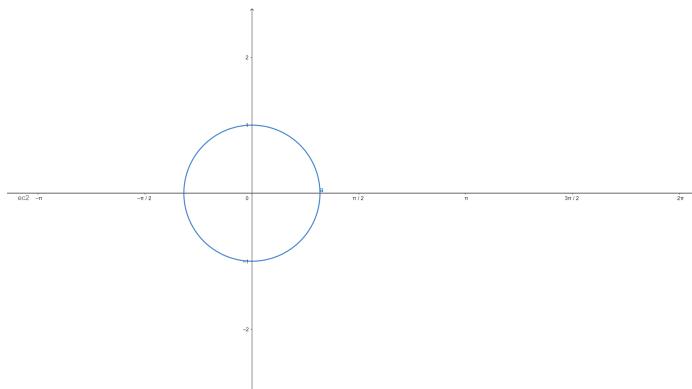


Figura 19: Curva paramétrica continua del tiempo sobre el círculo unitario

GeoGebra: Generación de la curva mediante las expresiones:

$$x(t) = \cos(2\pi * t / 86400), \quad y(t) = \sin(2\pi * t / 86400), \\ \text{Curve}(x(t), y(t), t, 0, 86400)$$

La figura muestra una curva paramétrica continua que representa el tiempo sobre el círculo unitario mediante funciones trigonométricas. Cada instante se proyecta como un punto único definido por $x(t) = \cos(\frac{2\pi t}{86400})$ y $y(t) = \sin(\frac{2\pi t}{86400})$, donde t es el número de segundos desde las 00:00. Esta codificación garantiza continuidad angular, permitiendo que modelos de simulación o aprendizaje automático interpreten correctamente la naturaleza cíclica del tiempo sin ambigüedades en los extremos del día.

Apéndice B. Algoritmos de los distintos agentes

```
1 Algoritmo Agente Isolation Forest
2
3 Agente_IsolationForest()
4     LeerConfiguración() // tasas, umbrales, ventanas
5     InicializarEstado() // colas, flags, historial
6
7 Escuchar()
8     Suscribir(canal_audio)
9     Mientras true
10        msg ← LeerMensaje()
11        Si msg válido (texto + timestamp)
12            evento ← NormalizarEvento(msg) // agrega date y label
13            Agregar(buffer, evento)
14            Si |buffer| > MAX_BUFFER
15                QuitarMasAntiguo(buffer)
16            EsperarBreve()
17
18 Procesar()
19     Mientras true
20        Si |buffer| > 0
21            lote ← Copia(buffer)
22            df ← ConvertirADataFrame(lote)
23            X ← GenerarCaracterísticas(df)
24            resultados ← Predecir(Modelo, X) // etiquetas y puntajes
25            anomalías ← Filtrar(resultados, pred = -1)
26            Para cada label único en anomalías
27                ConstruirMensaje(label, fecha, score)
28                Publicar(canal_alertas, mensaje)
29    EsperarIntervaloProceso()
```

Figura 20: Algoritmo del agente de detección de anomalías basado en Isolation Forest



```
1 Algoritmo Agente LSTM
2
3 Agente_LSTM()
4     LeerConfiguración() // tasas, umbrales, ventanas
5     InicializarEstado() // colas, flags, historial
6
7 Escuchar()
8     Suscribir(canal_entrada)
9     Mientras true
10        msg ← LeerMensaje()
11        Si msg es válido (tiene texto y timestamp)
12            evento ← ConstruirEvento(msg)
13            Agregar(buffer, evento)
14            Si |buffer| > SEQ_LEN
15                EliminarMásAntiguo(buffer)
16            EsperarBreve()
17
18 Analizar()
19     Mientras true
20        Si |buffer| ≥ SEQ_LEN
21            serie ← Copia(buffer)
22            Para cada evento en serie
23                CalcularFechaYHora()
24                DerivarCaracterísticasCíclicas(día, tiempo)
25            labels_enc ← CodificarEtiquetas(serie.labels)
26            X ← Combinar(labels_enc, características_temporales)
27            X2 ← ReconstrucciónModelo(X)
28            mse ← ErrorCuadráticoMedio(X, X2)
29            Si mse > umbral
30                alerta ← ConstruirAlerta(serie, mse)
31                Publicar(canal_alertas, alerta)
32                Limpiar(buffer)
33            EsperarIntervaloCorto()
```

Figura 21: Algoritmo del agente de predicción basado en LSTM

```
1 Algoritmo Agente_Silencio
2
3 Agente_Silencio( )
4   LeerConfiguración() // tasas, umbrales, ventanas
5   InicializarEstado() // colas, flags, historial
6   IniciarStreamAudio(callback_audio)
7
8 CapturaYProcesamientoAudio( )
9   Mientras true
10     frame ← ObtenerDeCola(timeout)
11     variance ← CalcularVarianza(frame)
12     actividad ← variance > UMBRAL
13     ActualizarUltimaActividad(actividad)
14     AñadirHistorial(frame, variance, actividad)
15     RecortarHistorialSiExcedeVentana( )
16     Si HistorialInsuficiente( )
17       Continuar
18     recent ← Últimos(STABILITY_FRAMES)
19     activity_count ← ContarActividad(recent)
20     silence_count ← STABILITY_FRAMES - activity_count
21     frames_sugieren_actividad ← activity_count > silence_count
22     hold ← TiempoDesdeUltimaActividad() ≤ HOLD_TIME
23     estado_actual_actividad ← frames_sugieren_actividad 0 hold
24     Si CambioASilencio(estado_actual_actividad)
25       PublicarEstado("silence", métricas)
26     Si CambioAAActividad(estado_actual_actividad)
27       PublicarEstado("activity", métricas)
28
29 PublicarEstado(tipo, datos)
30
31 Fin
```

Figura 22: Algoritmo del agente de detección de silencio

Apéndice C. Verificación del sistema de monitoreo acústico

Requerimiento	Descripción	Entrada	Salida	Criterio	de
				Aceptación	
R1	Captura continua del audio ambiental a través de micrófonos.	Señales de audio del entorno	Eventos discretos a etiquetados	El sistema debe estar activo y registrando datos	en todo momento.
R2	Procesamiento del audio capturado para su caracterización.	Flujo de datos de audio.	Clasificación del sonido (ej. voz, silencio, golpe).	El sistema debe etiquetar correctamente la señal de audio que recibe en todo momento.	
R3	Caracterizar el perfil de la actividad acústica "típica" del entorno, basado en la estacionalidad de los eventos sonoros clasificados.	Secuencia de eventos sonoros clasificados	Perfil acústico del entorno	El sistema construye un modelo del patrón de comportamiento sonoro habitual normal del entorno considerando variaciones temporales y estacionalidad.	

Requerimiento	Descripción	Entrada	Salida	Criterio de Aceptación
R4	procesar y comparar la actividad sensada de manera inmediata con el perfil de normalidad para detectar patrones anómalos con la mínima posible	Identificación de anomalía o evento atípico.	Detectar desviaciones significativas del patrón normal con baja latencia para respuesta oportuna.	
R5	El sistema envía notificaciones de emergencia si la anomalía es crítica o el usuario no responde.	Falta de respuesta del usuario o gravedad de la anomalía	Envío de notificaciones.	El sistema es capaz de enviar una alerta sin la intervención del usuario.

Tabla 5: Requerimientos del sistema acústico

Apéndice D. Cuadro comparativo de microcontroladores

Característica	Raspberry Pi	NVIDIA	Google Coral	Arduino	(ej. UNO/Mega)	ESP32
	4 model B	Jetson Nano	(TPU)			
Capacidad de Cómputo	Procesador ARM de alto rendimiento multinúcleo.	Procesador ARM de cuatro núcleos su GPU y NVIDIA CUDA.	Procesador ARM, pero su poder de 128 núcleos de una TPU NVIDIA especializada.	Microcontrolador de 8 o 32 bits.	Microcontrolador de 32 bits con Wi-Fi y Bluetooth.	
Memoria y almacenamiento	2-8 GB RAM, microSD hasta 1 TB	4 GB RAM, microSD hasta 128 GB	1 GB RAM, microSD, almacenamiento interno	2-8 KB RAM, almacenamiento persistente	512 KB RAM, soporte microSD externo	KB
Costo	65\$ (Amazon)	165\$ (AliBaba)	144\$	27\$	Entre 10\$ y 20\$ (Amazon)	

Tabla 6: Cuadro comparativo de microcontroladores

Apéndice E. Manual de Usuario

Introducción

Este sistema es una solución basada en inteligencia artificial, diseñada para identificar sonidos ambientales y generar alertas en situaciones de emergencia que puedan indicar que una persona se encuentra en una posición de vulnerabilidad.

El objetivo principal del sistema es ofrecer un monitoreo continuo, discreto y no invasivo, capaz de detectar eventos sonoros anómalos (gritos de auxilio, caídas, rotura de cristales o silencios prolongados) y notificar de manera inmediata a contactos de emergencia predefinidos.

El sistema integra tres componentes clave:

- Un módulo de clasificación acústica que utiliza modelos preentrenados como YAMNet y Vosk para identificar sonidos y palabras clave.
- Un módulo de detección de anomalías basado en modelos de inteligencia artificial como LSTM e Isolation Forest, que analiza secuencias de sonidos para identificar comportamientos atípicos.
- Un módulo de notificaciones que envía alertas en tiempo real a través de Telegram, correo electrónico o SMS.

Este manual contiene las instrucciones necesarias para interactuar con el sistema, comprender sus señales y responder adecuadamente en diferentes situaciones.

Desarrolladores del sistema

El Sistema de Monitoreo Acústico fue desarrollado por los tesis Carmelo Naim y César Sotillo, estudiantes de Ingeniería Informática de la Universidad Católica Andrés Bello, como parte de su Trabajo de Grado, realizado entre 2024 y octubre de 2025.

Requisitos previos

Antes de comenzar a utilizar el sistema, asegúrese de cumplir con los siguientes requisitos:

Conocimiento previo:

- No se requieren conocimientos técnicos avanzados.
- Es recomendable familiarizarse con los comandos de voz que el sistema reconoce.

Requisitos del sistema:

- Hardware: Dispositivo Raspberry Pi 4, micrófonos USB, conexión estable a internet y energía eléctrica continua
- Software: Aplicación de mensajería Telegram instalada en el teléfono del usuario o contacto designado.
- Red: Conexión Wi-Fi o Ethernet.

Requisitos previos

El sistema opera de manera autónoma y no requiere una interfaz gráfica para su funcionamiento cotidiano. Sin embargo, el usuario puede interactuar con él mediante comandos de voz y recibir notificaciones.

Componentes principales:

- Agentes de audio: Programas que escuchan continuamente el ambiente y clasifican sonidos.
- Sistema de alertas: Envía notificaciones automáticas cuando detecta una posible emergencia.
- Interacción por voz: Permite al usuario cancelar alertas o confirmar su estado.

Flujo de interacción:

- El agente de audio escucha continuamente el ambiente.

- Al detectar un sonido clasifica.
- Si se identifica una anomalía, se activa el sistema de alertas.
- El usuario recibe una notificación.
- El usuario puede interactuar por voz para cancelar la alerta y confirmar su estado.

Problemas comunes

A continuación, se presentan algunos problemas que pueden surgir durante el uso del sistema y sus posibles soluciones:

1. El sistema no responde a comandos de voz

- Causa: Ruido ambiental alto o micrófono obstruido.
- Solución: Hable en un tono claro y dirigido hacia el micrófono. Reduzca el ruido de fondo si es posible.

2. Falsos positivos

- Causa: El sistema puede estar interpretando eventos comunes como anomalías.
- Solución: Cancela la alerta usando el comando de voz “Estoy bien”.

3. No se reciben notificaciones en Telegram

- Causa: Problemas de conexión a Internet.
- Solución: Verifique la conexión a Internet.

Notas importantes

- Asegúrese de que el micrófono esté correctamente conectado y configurado en su dispositivo. Además de estar colocado en un lugar adecuado para captar el sonido.
- Mantenga el software del sistema actualizado el software del sistema para mejorar la precisión en la detección.

- El sistema no almacena grabaciones de audio. Todo el procesamiento se realiza de forma local y en tiempo real.

Apéndice F. Manual de Sistema

Introducción

El siguiente documento es la guía de instalación de Alertify, un sistema de monitoreo acústico inteligente diseñado para la identificación de sonidos ambientales y la generación automática de alertas ante eventos sonoros anómalos que puedan indicar una situación de emergencia o vulnerabilidad.

El sistema está especialmente orientado a entornos domésticos o asistidos, proporcionando una solución no invasiva para el monitoreo continuo de personas que puedan requerir asistencia inmediata, como adultos mayores, personas con discapacidad o cualquier individuo en situación de riesgo.

La arquitectura del sistema se compone de los siguientes módulos principales:

1. **Agentes de captura y detección**, que ejecutan modelos especializados de inteligencia artificial para:

- Reconocimiento de voz en tiempo real (Vosk)
- Clasificación de eventos acústicos (YAMNet)
- Detección de anomalías en secuencias sonoras (LSTM, Transformer, Isolation Forest)
- Monitoreo de períodos de silencio prolongado (Silents Agent)
- Persistencia de eventos (Data Agent)
- Gestión de notificaciones de emergencia (Emergency Agent)

La arquitectura está realizada para operar en dispositivos de edge computing como el Raspberry Pi 4 modelo B, garantizando que el procesamiento de audio se realice de forma local y privada, sin almacenar grabaciones ni depender de servicios en la nube para el análisis sensible.

Este manual está dirigido a administradores de sistemas y usuarios técnicos, y proporciona instrucciones completas para la instalación, configuración, operación y resolución de problemas.

Requisitos

Hardware

1. Raspberry Pi 4 Modelo B con las siguientes especificaciones:

- Procesador: Broadcom BCM2711, SoC de cuatro núcleos Cortex-A72 (ARM v8) de 64 bits a 1,8 GHz
- Memoria Ram: 4GB
- Almacenamiento: MicroSD 32GB Clase 10 o superior
- Puertos: 2 × USB 3.0, 2 × USB 2.0, 2 × micro-HDMI
- Audio: Entrada de micrófono via USB
- Microfonos: Adafruit Mini Microfono USB 3367

Software

1. Raspberry Pi 4 Modelo B:

- Sistema operativo: Raspberry Pi OS (64-bit) Bullseye o superior
- Python: Versión 3.9 o 3.10
- Librerías a utilizar: pandas, joblib, scikit-learn, tensorflow, numpy, sounddevice, python-socketio, pyaudio, vosk, aiohttp, psycopg2, aioredis, pyttsx3

Infraestructura de red

1. Puertos requeridos

- Redis: 6379

2. Conectividad

- Red local estable entre Raspberry Pi y servidor
- Acceso a internet para notificaciones externas (Telegram, SMTP)

Modelo de IA y recursos.

1. Modelos Preentrenados

- Yamnet: yamnet-agent/yamnet.tflite, yamnet-agent/yamnet_class_map.csv
- VOSK (español): vosk-agent/vosk-model-small-es-0.42/
- Modelo de Anomalías:
 - a) lstm-agent/lstm_autoencoder.keras y lstm-agent/umbral.pkl

2. Conectividad

- Red local estable entre Raspberry Pi y servidor
- Acceso a internet para notificaciones externas (Telegram, SMTP)

Instalación

Cliente (alertify-client).

1. Clonar y abrir carpeta Alertify-client.
2. Verificar Python 3.9+ y crear un entorno virtual (opcional).
3. Asegurar que los siguientes modelos estén en sus rutas correspondientes.
4. Ejecutar los agentes y scripts principales

Configuración

Redis

Los agentes usan por defecto redis://default:alertify@localhost:6379. Si cambia el host o las credenciales, actualice los scripts o variables de entorno correspondientes.

Configuración de audio

En silents-agent/silents-agent.py, verificar y ajustar según sea necesario:

- SAMPLE_RATE
- VARIANCE_THRESHOLD
- Otros parámetros relacionados con la captura de audio

Carpetas y Archivos (alertify-client).

Archivos en la raíz

Componentes principales para orquestar y documentar el cliente.

Archivo	Descripción
main.py	Lanza los agentes como procesos independientes, maneja señales de parada y espera su finalización.
README.md	Introducción y notas del cliente
requirements.txt	Lista de dependencias Python necesarias para ejecutar los agentes (pandas, tensorflow, aioredis, vosk, etc.).
.gitignore	Reglas de exclusión de Git (archivos temporales, modelos, entornos, etc.).

CSV

Catálogos en CSV utilizados por agentes de audio para mapear clases y categorías.

Archivo	Descripción
Clasificación de categorías.csv	Tabla de categorías de eventos/sonidos normalizados para etiquetado y análisis.
yamnet_class_map.csv	Etiqueta legible de YAMNet usado por el agente de clasificación.

data_agent

Ingesta de eventos desde Redis, persistencia en PostgreSQL y generación periódica de dataset

Archivo	Descripción
config.py	Lee data_agent/database.ini con ConfigParser y devuelve un dict de conexión PostgreSQL.
data_agent.py	Suscribe a Redis (alertas_audio, alertas_vosk), inserta eventos en tabla event y actualiza data/dataset.csv periódicamente.
generate_csv.py	Genera data/dataset.csv on-demand; normaliza zona horaria (America/Caracas) y guarda fechas sin tz.
postgres_connection.py	Gestiona una conexión global a PostgreSQL con psycopg2 usando la configuración cargada.

emergency_agent

Agente que gestiona las alertas de emergencia.

Archivo	Descripción
emergency_agent.py	Se suscribe a múltiples canales Redis, agrega alertas, permite pausar/reanudar, hace TTS y envía notificaciones a Telegram; expone control por vosk_commands / emergency_control.

Isolation-forest-agent

Detección de anomalías de secuencias de eventos con Isolation Forest.

Archivo	Descripción
encoder.pkl	Codificador para transformar etiquetas de eventos a vectores.
isolation-forest-agent.py	Escucha alertas_audio, construye features temporales (día/tiempo cíclico), detecta anomalías por batch y publica en alertas_isolation_forest; si falta modelo, entrena y guarda anomalies_model.pkl.

lstm-agent

Detección de anomalías con Autoencoder LSTM en ventanas temporales

Archivo	Descripción
encoder.pkl	Codificador para transformar etiquetas a vectores de entrada del modelo.
lstm_autoencoder.keras	Modelo Keras de Autoencoder LSTM.
lstm-agent.py	Consumo alertas_audio, arma secuencias (longitud fija), calcula MSE de reconstrucción y publica series anómalas en alertas_lstm si supera el umbral.
umbral.pkl	Umbral de MSE recomendado para lstm_autoencoder.keras.

silent-agent

Detección en tiempo real de silencio/actividad a partir de audio del micrófono.

Archivo	Descripción
silents-agent.py	Captura audio, calcula varianza por frame, aplica ventana de estabilidad y periodo de retención, y publica cambios de estado (silencio/actividad) en alertas_silencio con métricas.

Vosk-agent

Reconocimiento de voz offline con Vosk en español y disparo de eventos por palabras clave.

Archivo	Descripción
vosk-agent.py	Escucha micrófono con sounddevice y Vosk, detecta palabras clave (auxilio/Estoy bien), publica en alertas_vosk y envía comandos a vosk_commands.

Vosk-model-small-es-0.42

Archivo	Descripción
README	Información del modelo y notas del proyecto Vosk.
am/final.mdl	Modelo acústico entrenado (acoustic model).
conf/mfcc.conf	Configuración de extracción de características MFCC.
conf/model.conf	Configuración general del modelo.
graph/disambig_tid.int	Archivo de desambiguación para el grafo de decodificación.
graph/Gr.fst	Grafo FST principal para decodificación.
graph/HCLr.fst	Composición HCL para el decodificador (fonética/lexicón).
graph/phones/word_bounday.int	Límites de palabras por audio.
ivector/final.dubm	UBM para i-vectors.
ivector/final.ie	Extracción de i-vectors.
ivector/final.mat	Matriz de proyección para i-vectors.
ivector/global_cmvn.stats	Estadísticas para normalización CMVN global.
ivector/online_cmvn.conf	Configuración CMVN online.
ivector/splice.conf	Configuración de splicing de características.

Yamnet-agent

Clasificación de audio en streaming con YAMNet (TensorFlow Lite) y publicación de eventos.

Archivo	Descripción
yamnet_class_map.csv	Archivo con el nombre de cada clase para interpretar predicciones.
yamnet-agent.py	Captura audio, infiere con TFLite, marca emergencias por índices críticos y publica eventos en alertas_audio.
yamnet.tflite	Modelo YAMNet en formato TensorFlow Lite usado por el agente.

Carpetas y Archivos (alertify-server).

Archivos en la raíz

Archivos base de configuración, tooling y metadatos del proyecto NestJS/TypeORM.

Archivo	Descripción
package.json	Scripts de desarrollo/producción (Nest, TypeORM, seeds), dependencias y configuración de Jest.
pnpm-lock.yaml	Bloqueo de dependencias (pnpm).
docker-compose.yml	Servicios contenedezados (p.ej. PostgreSQL/Redis si aplica) para desarrollo.
nest-cli.json	Configuración del CLI de Nest (paths de compilación).
tsconfig.json	Configuración TypeScript principal.
tsconfig.build.json	Configuración TypeScript para compilación a producción (dist).
.eslintrc.js	Reglas de ESLint.
.prettierrc	Reglas de formateo Prettier.
.gitignore	Exclusiones de Git.
README.md	Documentación del servidor.

SRC

Código fuente del servidor (NestJS), organizado por módulos y configuración.

Archivo	Descripción
main.ts	Inicializa contexto transaccional, configura Swagger en /docs, prefijo api/v1 y levanta el servidor en PORT.
app.module.ts	Carga .env, registra TypeORM y el módulo Events, habilita @nestjs/schedule.
config/	Configuración de entorno y base de datos (TypeORM, DataSource, variables).
modules/	Módulos de funcionalidad (actualmente events).

src/config

Configuraciones de entorno y base de datos

Archivo	Descripción
environment.ts	Cargar variables desde .env y las exporta como objeto envConfig.
database.config.ts	Opciones DataSourceOptions para PostgreSQL: host/puerto/credenciales/DB, entities, synchronize, migraciones y timezone.
database.ts	Vuelve a exportar propiedades básicas de conexión a DB a partir de environment.
datasource.config.ts	Instancia DataSource (TypeORM) desde database.config para migraciones/CLI.
typeorm.config.ts	Config asíncrono de TypeORM para Nest; integra typeorm-transactional con addTransactionalDataSource.

src/modules/events

Módulo responsable de WebSockets (Socket.IO), gestión de eventos, emergencias, actividad y configuración de alarmas.

Archivo	Descripción
events.gateway.ts	Gateway WebSocket: maneja conexión/desconexión, comandos imOk, help-call, alarmOn/Off, emite emergency-alert y reproduce TTS (voz).
events.service.ts	Lógica de negocio: CRUD de eventos/emergencias, cron jobs para silencios y emergencias, y notificaciones (Telegram/Email/TTS).
enums/events.enums.ts	Enumeración Calls para tipos de llamadas (imOk, helpCall).

Links a los repositorios

1. Alertify-client: <https://github.com/Cariea/alertify-client>
2. Alertify-server: <https://github.com/Cariea/alertify-server>

Anexos

Anexo A. Dispositivo Raspberry Pi 4 Model B

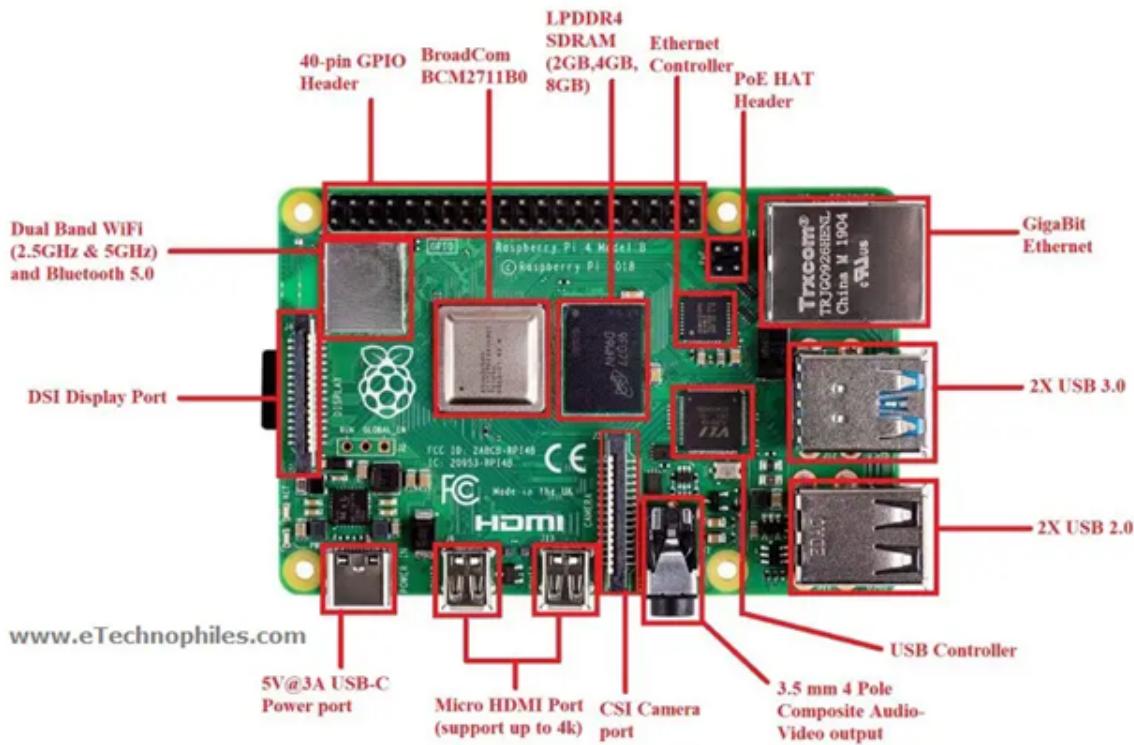


Figura 23: Arquitectura del Raspberry Pi 4 Model B

Anexo B. Construcción del Isolation Forest (Bosque de Aislamiento)

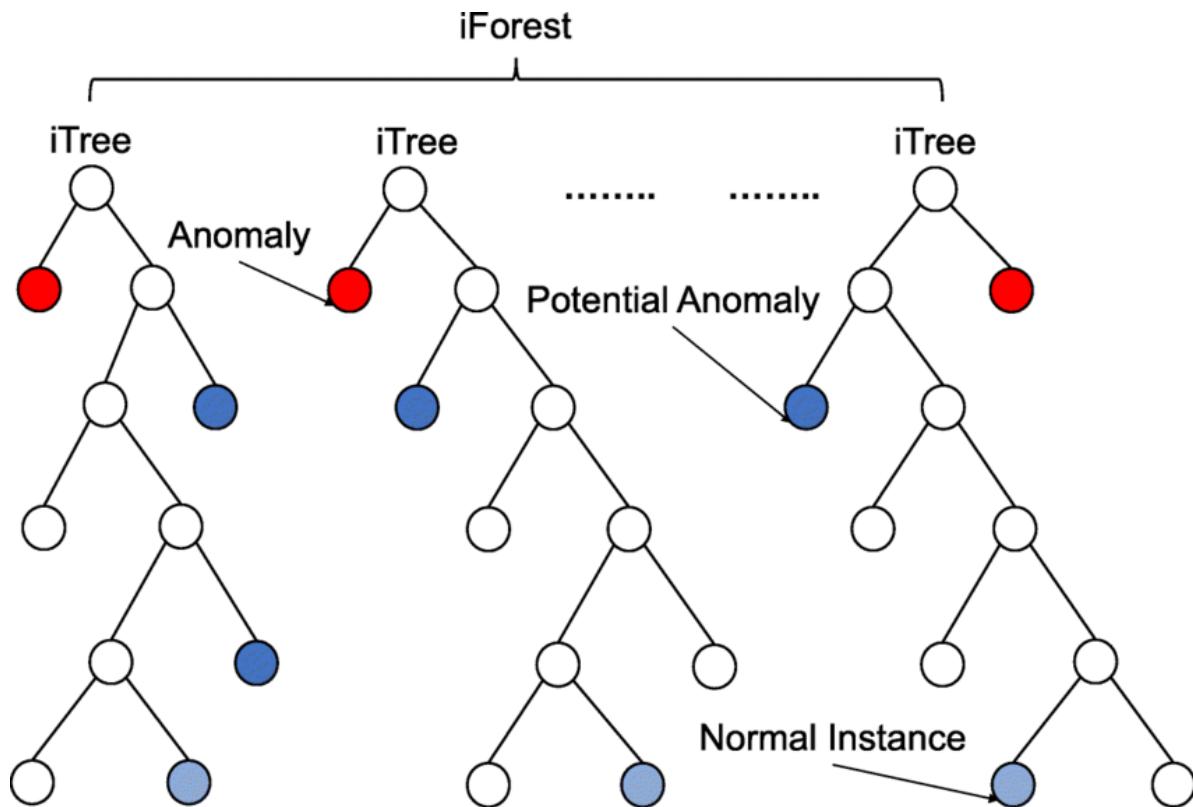


Figura 24: Algoritmo de construcción del Isolation Forest

Anexo C. Número de datos de entrenamiento por categoría de sonido

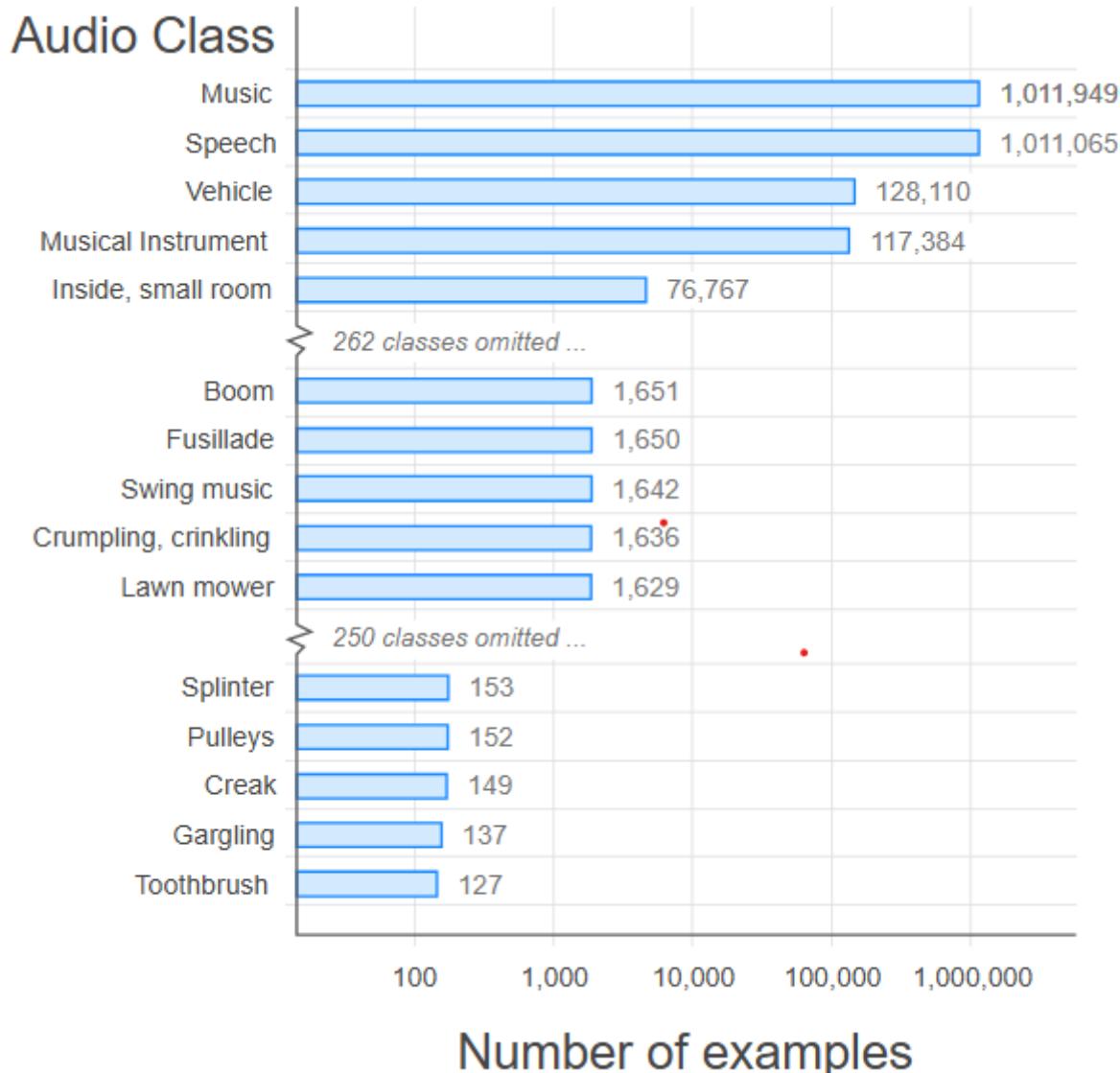


Figura 25: Número de datos de entrenamiento por categoría de sonido

Anexo D. Categorías de sonido del modelo YAMNet

Tabla 21: Lista de Índices y Categorías de Sonido

Índice	Categoría
0	Hablar
1	Niño hablando; bebe hablando
2	Conversacion
3	Narracion; monologo
4	Balbuceo
5	sintetizador de voz
6	Vociferar
7	Bramido
8	Alarido
9	Gritar
10	Niños vociferando
11	Grito
12	Susurro
13	Risa
14	Bebe riendo
15	Risilla
16	Risa disimulada
17	Carcajada
18	Risa moderada
19	Llorando; sollozando
20	Llanto de bebe; llanto de infante

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
21	Gemido
22	Llorar; gemir
23	Suspiro
24	Cantando
25	Coro
26	Tiroles
27	Corear
28	Mantra
29	Niños cantando
30	canto sintetico
31	Golpecitos
32	Tarareo
33	Quejido
34	Gruñido
35	Silbido
36	Respiracion
37	Resuello
38	Ronquidos
39	Resoplido
40	Jadear
41	Bufido
42	Tos
43	Despejar la garganta
44	Estornudo
45	Oler
46	Correr

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
47	Barajar
48	Caminar; pasos
49	Masticar; mordisquear
50	Morder
51	Gargarizar
52	Ruido estomacal
53	Eructos; eructar
54	Hipo
55	Pedo
56	Manos
57	Chasquear los dedos
58	Aplaudir
59	Sonidos del corazon; latidos del corazon
60	Soplo cardiaco
61	Animar
62	Ovacion
63	Charla
64	Multitud
65	Alboroto; ruido del habla; balbuceo del habla
66	Niños jugando
67	Animal
68	Animales domesticos; mascotas
69	Perro
70	Ladrar
71	Chillido
72	Aullido

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
73	Guau-guau
74	Gruñendo
75	Gemido (perro)
76	Gato
77	Ronroneo
78	Miau
79	Siseo
80	Chillar
81	Ganado; animales de granja; animales de trabajo
82	Caballo
83	Clip-clop
84	Relincho; Relinchito
85	Ganado bovino
86	Muu
87	Cencerro
88	Cerdo
89	Oink
90	Cabra
91	Baa (Cabra; Oveja)
92	Oveja
93	Ave de corral
94	Pollo; gallo
95	Cacareo
96	Cacareo; quiquirikii
97	Pavo
98	Gluglu

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
99	Pato
100	Cuac
101	Ganso
102	Graznido
103	Animales Salvajes
104	Gatos rugiendo (leones; tigres)
105	Rugido
106	Pájaro
107	Vocalizacion de pájaros; canto de pájaros; canto de pájaros
108	Trino
109	Graznido
110	Paloma; tórtola
111	Coo
112	Cuervo
113	Caw
114	Búho
115	Hoot
116	Vuelo de pájaro; batir de alas
117	Caninos; perros; lobos
118	Roedores; ratas; ratones
119	Raton
120	Golpeteo
121	Insecto
122	Grillo
123	Mosquito
124	Mosca; Mosca doméstica

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
125	Zumbido
126	Abeja; avispa; etc.
127	Rana
128	Croar
129	Serpiente
130	Cascabel
131	Vocalizacion de ballenas
132	Música
133	Instrumento musical
134	Instrumento de cuerda pulsada
135	Guitarra
136	Guitarra eléctrica
137	Bajo
138	Guitarra acústica
139	Guitarra de acero; guitarra slide
140	Tapping (técnica de guitarra)
141	Rasguear
142	Banjo
143	Sitar
144	Mandolin
145	Zither
146	Ukulele
147	Teclado (musical)
148	Piano
149	Piano eléctrico
150	Órgano

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
151	Órgano eléctrico
152	Órgano de hammond
153	Sintetizador
154	Sampler
155	Clavecin
156	Percusión
157	Batería
158	Caja de ritmos
159	Tambor
160	Caja (instrumento)
161	Rimshot
162	Redoble de tambores
163	Bombo
164	Tímpanos
165	Tabla
166	Platillo
167	Hi-hat
168	Bloque de madera
169	Pandereta
170	Sonajero (instrumento)
171	Maraca
172	Gong
173	Campanas tubulares
174	Percusión de mazo
175	Marimba; xilófono
176	Campanas

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
177	Vibrófono
178	Cacerola
179	Orquesta
180	instrumento de latín
181	Corno francés
182	Trompeta
183	Trombón
184	Instrumento de cuerda frotada
185	Sección de cuerdas
186	Violín; violín tradicional
187	Pizzicato
188	Violonchelo
189	Contrabajo
190	Instrumento de viento; instrumento de viento de madera
191	Flauta
192	Saxofón
193	Clarinete
194	Arpa
195	Campana
196	Campana de la iglesia
197	Cascabel
198	Timbre de bicicleta
199	Diapasón
200	Campanilla
201	Carillón de viento
202	Cambio de repique (campanología)

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
203	Armónica
204	Acordeón
205	Gaita
206	Didyeridú
207	Shofar
208	Theremin
209	Cuenco tibetano
210	Scratching (movimiento de discos)
211	Música pop
212	Música hip hop
213	Beatboxing
214	Música rock
215	Heavy metal
216	Punk rock
217	Grunge
218	Rock progresivo
219	Rock and roll
220	Rock psicodélico
221	ritmo y blues
222	música soul
223	Reggae
224	País
225	Música swing
226	pasto azul
227	Canguelo
228	Música folklórica

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
229	música del medio oriente
230	Jazz
231	Disco
232	Música clásica
233	ópera
234	música electrónica
235	música house
236	tecnología
237	dubstep
238	batería y bajo
239	electrónica
240	música electrónica de baile
241	música ambiental
242	música trance
243	música de latinoamérica
244	música salsa
245	Flamenco
246	Blues
247	música para niños
248	Música de la nueva era
249	Música vocal
250	a capella
251	Música de África
252	afrobeat
253	música cristiana
254	música gospel

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
255	Música de Asia
256	música carnática
257	Música de Bollywood
258	ska
259	música tradicional
260	música independiente
261	Canción
262	Música de fondo
263	tema musical
264	Jingle (música)
265	Música de banda sonora
266	Canción de cuna
267	música de videojuegos
268	música navideña
269	música de baile
270	música de boda
271	música feliz
272	música triste
273	música tierna
274	música emocionante
275	música enojada
276	música aterradora
277	Viento
278	hojas susurantes
279	Ruido del viento (micrófono)
280	Tormenta

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
281	Trueno
282	Agua
283	Lluvia
284	Gota de agua
285	Lluvia en superficie
286	Arroyo
287	Cascada
288	Océano
289	olas; surf
290	Vapor
291	Gluglu
292	Fuego
293	Crepitar
294	Vehículo
295	Barco; Vehículo acuático
296	Velero; velero
297	Bote de remos; canoa; kayak
298	Lancha motora; lancha rápida
299	Barco
300	Vehículo de motor (carretera)
301	Auto
302	Bocina de vehículo; bocina de coche; bocina
303	Sonar
304	alarma de coche
305	elevalunas eléctricos; elevalunas eléctricos
306	derrapando

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
307	Chirrido de neumáticos
308	Coche pasando por
309	Coche de carreras; carreras de autos.
310	Camión
311	Freno de aire
312	Bocina de aire; bocina de camión
313	Pitidos de marcha atrás
314	Camión de helados; furgoneta de helados.
315	Autobús
316	vehículo de emergencia
317	Coche de policía (sirena)
318	Ambulancia (sirena)
319	Camión de bomberos; camión de bomberos (sirena)
320	Motocicleta
321	Ruido del tráfico; ruido de la carretera.
322	Transporte ferroviario
323	Tren
324	silbato de tren
325	Bocina de tren
326	Vagón de ferrocarril; vagón de tren.
327	Ruedas de tren chirriando
328	Metro; metro; metro
329	Aeronave
330	motor de avión
331	motor a reacción
332	Hélice

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
333	Helicóptero
334	Avión de ala fija; avión.
335	Bicicleta
336	Monopatín
337	Motor
338	Motor ligero (alta frecuencia)
339	Taladro dental; taladro de dentista.
340	Cortacésped
341	motosierra
342	Motor medio (frecuencia media)
343	Motor pesado (baja frecuencia)
344	Golpeteo del motor
345	Arranque del motor
346	De marcha en vacío
347	Acelerando; acelerando; vroom
348	Puerta
349	Timbre de la puerta
350	ding dong
351	Puerta corrediza
352	Golpe
353	Golpear
354	Grifo
355	Chirrido
356	Armario abierto o cerrado
357	Cajón abierto o cerrado
358	Platos; ollas y sartenes

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
359	cubiertos; cubiertos
360	Picar (comida)
361	Freír (comida)
362	Microondas
363	Licuadora
364	Grifo de agua; grifo
365	Fregadero (llenado o lavado)
366	Bañera (llenado o lavado)
367	Secador de pelo
368	Descarga del inodoro
369	Cepillo de dientes
370	cepillo de dientes eléctrico
371	Aspiradora
372	Cremallera (ropa)
373	Llaves tintineando
374	Moneda (cayendo)
375	Tijeras
376	Afeitadora eléctrica; afeitadora eléctrica.
377	barajar cartas
378	Mecanografía
379	Máquina de escribir
380	teclado de computadora
381	Escribiendo
382	Alarma
383	Teléfono
384	Suena el timbre del teléfono

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
385	Tono de llamada
386	Marcación telefónica; DTMF
387	Tono de marcación
388	señal de ocupado
389	Despertador
390	Sirena
391	sirena de defensa civil
392	Zumbador
393	Detector de humo; alarma de humo.
394	Alarma de incendios
395	Sirena
396	Silbar
397	silbato de vapor
398	Mecanismos
399	Trinquete; trinquete
400	Reloj
401	Garrapata
402	tic-tac
403	Engranajes
404	poleas
405	Máquina de coser
406	ventilador mecánico
407	Aire acondicionado
408	Caja registradora
409	Impresora
410	Cámara

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
411	Cámara réflex de un solo objetivo
412	Herramientas
413	Martillo
414	Martillo neumático
415	Aserradura
416	Limar (escofina)
417	Lijado
418	Herramienta eléctrica
419	Perforar
420	Explosión
421	Disparo; disparos
422	Ametralladora
423	Descarga cerrada
424	Fuego de artillería
425	pistola de gorra
426	Fuegos artificiales
427	Petardo
428	Explosión; estallido
429	Chin; tintineo
430	Líquido
431	Salpicar; salpicar
432	Pegar
433	Chapotear
434	Derramar
435	Goteo; regateo
436	Llenar (con líquido)

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
437	Erupción
438	Auge
439	Madera
440	Cortar
441	Astilla
442	Grieta
443	Vaso
444	Romper
445	Goteo
446	Chorro
447	Pulverización
448	Bomba (líquido)
449	Remover
450	Hirviendo
451	Sonar
452	Flecha
453	Whoosh; swoosh; silbido
454	Golpe; golpe
455	Tonk
456	sintonizador electrónico
457	Unidad de efectos
458	efecto coro
459	rebote de baloncesto
460	Estallido
461	Bofetada; bofetada
462	Golpe; golpe

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
463	Aplastar; chocar
464	Rotura
465	Fuerte
466	Látigo
467	Solapa
468	Rascar
469	Raspar
470	Frotar
471	Rollo
472	Aplastante
473	Arrugando; arrugando
474	lagrimeo
475	Bip; bip
476	Silbido
477	Timbre
478	Sonido metálico
479	Chillido
480	Crujir
481	Crujido
482	Zumbido
483	Estrepito
484	Chisporrotear
485	Haciendo clic
486	Clic-clac
487	Retumbar
488	Plaf

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
489	Jingle; tintineo
490	Tararear
491	Gusto
492	boing
493	Crujido
494	Silencio
495	onda sinusoidal
496	Armónico
497	tono de chirrido
498	efecto de sonido
499	Legumbres
500	Interior; habitacion pequeña.
501	Interior; gran salon o recibidor.
502	Interior; espacio publico.
503	Exterior; urbano o artificial
504	Exterior; rural o natural
505	Reverberacion
506	Eco
507	Ruido
508	Ruido ambiental
509	Estatico
510	Zumbido de red
511	Distorsion
512	efecto local
513	Cacofonia
514	ruido blanco

Continuación en la página siguiente

Cuadro 21 – Continuación

Índice	Categoría
515	ruido rosa
516	Palpitante
517	Vibracion
518	Television
519	Radio
520	Grabacion de campo