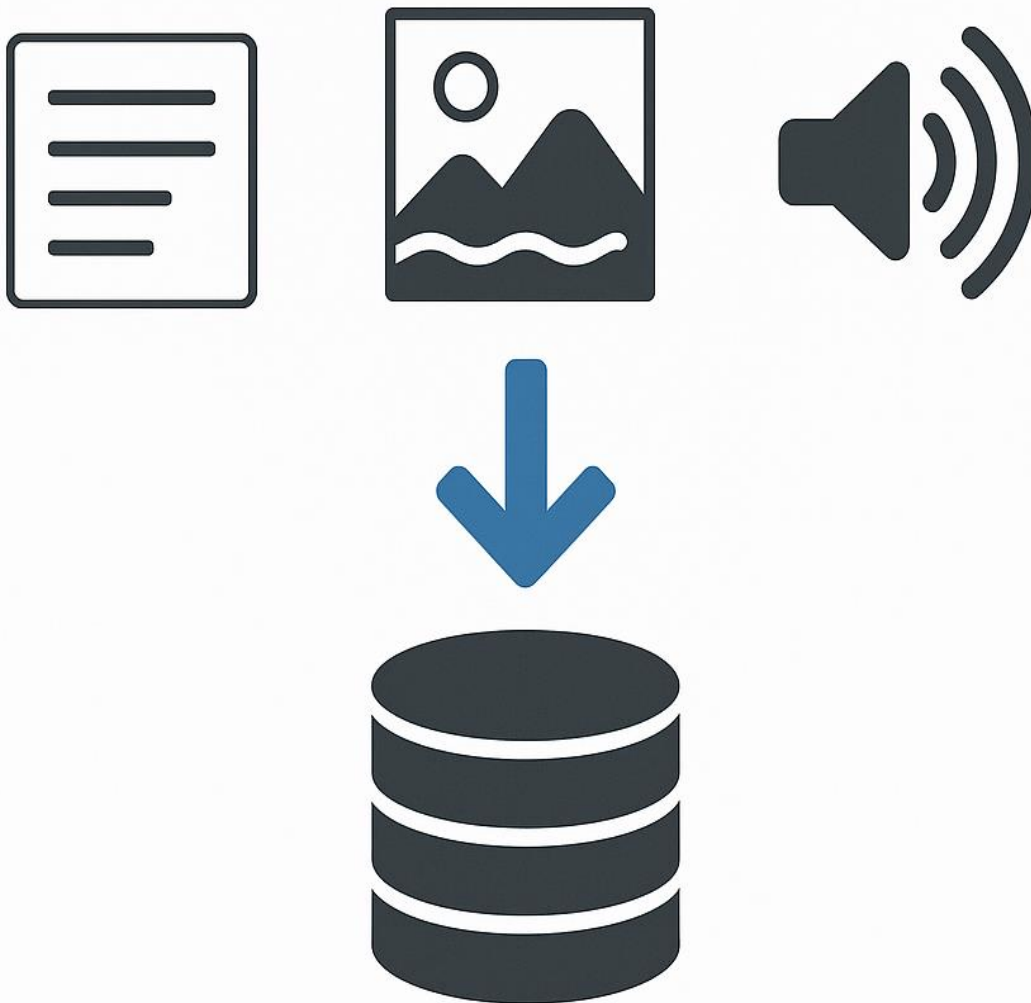


MODELOS DE LENGUAJE PARA LA GENERACIÓN DE DATASETS



CERTEX AI

Introducción

Estructurar correctamente la salida de los modelos es clave para su integración con sistemas reales. Este proceso comienza con la comprensión de los fundamentos de la estructuración multimodal, que abarca desde qué es un modelo de lenguaje, hasta cómo estos pueden interactuar con diferentes tipos de datos (texto, imágenes, audio) mediante la multimodalidad. La estructuración de salida, especialmente en formatos como JSON, CSV, tablas o XML, permite representar la información de manera precisa, jerárquica y reutilizable. Comprender por qué estructurar, y cómo los modelos de lenguaje pueden generar salidas organizadas, es fundamental para automatizar tareas, integrar con APIs, generar reportes o alimentar bases de datos. A través de esta base, se establece una transición natural hacia el control avanzado del formato, incluyendo técnicas como strip, function calling y outlines, que refuerzan la precisión y confiabilidad del modelo.

A partir de ahí, la estructuración se extiende más allá del texto, permitiendo extraer y representar conocimiento con prompts diseñados para convertir texto en JSON estructurado usando esquemas definidos, reconocimiento de entidades, y el uso de herramientas como JSON Schema y spaCy. En el plano visual, se explora cómo los modelos multimodales como GPT-4 Vision permiten transformar imágenes en datos estructurados mediante OCR y análisis semántico visual. Lo mismo aplica al audio: herramientas como Whisper y Gemini permiten convertir voz en texto y luego estructurarlo en formatos útiles. Finalmente, se discuten los desafíos específicos de cada modalidad (texto, imagen, audio), los errores comunes, y cómo manejar la incertidumbre en salidas no controladas, cerrando con una mirada práctica sobre herramientas adicionales como Pydantic, patrones de reintento, y otros proveedores que amplían el ecosistema de estructuración en IA.

Temario

1. Fundamentos de la estructuración multimodal

- 1.1. ¿Qué es un modelo de lenguaje?
- 1.2. ¿Qué es multimodalidad?
- 1.3. Salida estructurada
 - 1.3.1. Tipos: JSON, CSV, tabla, XML
- 1.4. ¿Por qué estructurar?
- 1.5. Salida estructurada con Modelos de lenguaje

2. Salida estructurada (JSON) y control de formato

- 2.1. ¿Qué es JSON Output?
 - 2.1.1. strip = true
 - 2.1.2. strip = false
- 2.2. Control del formato de salida
 - 2.2.1. Function calling
 - 2.2.2. Outlines
- 2.3. Salida estructurada, semiestructurada y no estructurada

3. Texto -> JSON estructurado con prompts y Json schema

- 3.1. Named Entity Recognition (NER)
- 3.2. Extracción de relaciones
- 3.3. Prompting para control estructural
- 3.4. JSON API
 - 3.4.1. Strict
 - 3.4.2. response_format=json
 - 3.4.3. functions
 - 3.4.4. json_mode
 - 3.4.5. schema

3.5. spaCy vs LLMs

4. Imagen -> Estructura

- 4.1. ¿Qué es OCR?
- 4.2. ¿Cómo combinar OCR con NLP?
- 4.3. GPT-4 Vision: ¿Cómo analiza imágenes y estructuras contenido?
- 4.4. OCR con LLMs
- 4.5. Convertir imagen en JSON: extracción tabular
 - 4.5.1. Salida estructurada desde imagen
 - 4.5.2. Salida estructurada con modelos no generativos

5. Comparación Multimodal: texto vs imagen vs audio

- 5.1. Dificultades de cada modalidad
- 5.2. Tipo de errores estructurales comunes
- 5.3. ¿Qué pasa cuando no se puede controlar la salida?

6. Audio -> Texto -> Estructura con Whisper y Gemini

- 6.1. ASR (Automatic Speech Recognition)
- 6.2. Whisper
 - 6.2.1. Variantes
 - 6.2.2. Chunks



- 6.2.3. Procesamiento BERT
- 6.2.4. Procesamiento LLM
- 6.3. Gemini
 - 6.3.1. Extracción Audio
 - 6.3.2. Opciones
 - 6.3.3. Costos
- 7. Extras**
 - 7.1. JSON API y pydantic
 - 7.2. Patrones de reintento
 - 7.3. Otras modalidades
 - 7.4. Otros proveedores



Recursos Necesarios

Conocimientos Básicos de:

- Windows.
- Editores de Texto.
- Python para la parte práctica (no necesario para la parte teórica).

1. Fundamentos de la estructuración multimodal

Los LLMs han transformado la forma en que interactuamos con la inteligencia artificial, permitiendo generar texto coherente, traducir, resumir y responder preguntas con alta precisión. Sin embargo, para que su uso sea verdaderamente útil en entornos productivos, se requiere que la información que generan no solo sea comprensible para humanos, sino también estructurada, predecible y fácilmente procesable por máquinas. En este contexto, la salida estructurada (especialmente en formatos como JSON, tablas o listas jerárquicas) se convierte en una pieza clave para integrar los LLM con APIs, bases de datos, dashboards y otros sistemas inteligentes. Además, en entornos multimodales donde se combinan texto, imagen y audio, formatos estructurados como JSON permiten organizar la información de forma clara, interoperable y lista para su análisis automático.

1.1. ¿Qué es un modelo de lenguaje?

Un modelo de lenguaje es un sistema de inteligencia artificial entrenado para comprender, procesar y generar texto en lenguaje natural de forma coherente y contextual. Estos modelos se construyen utilizando grandes volúmenes de texto, desde libros, artículos y páginas web, hasta conversaciones humanas, lo que les permite aprender patrones, relaciones semánticas, estructuras gramaticales y estilos lingüísticos. En esencia, un modelo de lenguaje puede predecir qué palabra o frase sigue a una secuencia dada, pero sus capacidades van mucho más allá.

1.2. ¿Qué es multimodalidad?

La multimodalidad es la capacidad de un sistema, especialmente en inteligencia artificial, para procesar e integrar múltiples tipos de información o “modalidades” al mismo tiempo, como texto, imágenes, audio, video, o incluso datos sensoriales (como señales biométricas o datos espaciales).

Un modelo multimodal no solo entiende texto (como lo haría un modelo de lenguaje tradicional), sino que también puede combinar diferentes tipos de entradas para razonar o generar contenido más complejo. Por ejemplo:

- Entender una pregunta escrita y responder basándose en una imagen adjunta.
- Describir una imagen usando texto (captioning).
- Leer texto en una imagen (OCR).
- Traducir el habla de un video mientras interpreta gestos faciales.

- Analizar un informe médico que combina texto clínico, radiografías e historiales.

Modalidades comunes en IA:

Modalidad	Tipo de dato que maneja	Ejemplos de tareas
Texto	Palabras, frases, párrafos	Resumen, traducción, clasificación
Imagen	Fotografías, dibujos, gráficos	Detección de objetos, captioning
Audio	Voz, música, sonidos	Reconocimiento de voz, emociones
Video	Secuencia de imágenes + audio	Análisis de escenas, seguimiento

1.3. Salida estructurada

Una salida estructurada es aquella en la que la información generada o extraída no se presenta como texto libre, sino en formatos organizados y predecibles, ideales para su análisis automatizado, almacenamiento o integración con otros sistemas.

A diferencia de una respuesta narrativa, una salida estructurada permite identificar claramente campos, categorías y relaciones entre los datos, lo que facilita su reutilización en sistemas informáticos.

1.3.1. Tipos de salida estructurada: JSON, CSV, tabla, XML

Los formatos más comunes de salida estructurada incluyen:

- **JSON (JavaScript Object Notation):**

JSON (JavaScript Object Notation) es el formato estructurado más común para intercambiar datos entre sistemas y modelos de lenguaje.

Características:

- Formato ligero y legible para representar datos estructurados en forma de clave-valor.
- Ideal para APIs, análisis de datos y bases de datos NoSQL.

Las **salidas estructuradas** en formato **JSON** permiten organizar la información de forma legible y útil para modelos, APIs y sistemas:

```
{
  "nombre": "Ana",
  "edad": 30,
  "ciudad": "Madrid"
}
```

- **CSV (Comma-Separated Values):**

El formato CSV (Comma-Separated Values) se utiliza principalmente para representar datos tabulares de forma simple, legible por humanos y fácil de importar a hojas de cálculo o bases de datos.

Características:

- Datos tabulares en texto plano, separados por comas (o punto y coma).
- Usado ampliamente para hojas de cálculo o importación/exportación de datos.

nombre,edad,ciudad

Ana,30,Madrid

- **Tabla:**

Una tabla es una forma visual y estructurada de representar información en filas y columnas. Aunque puede estar respaldada por formatos como CSV, HTML, Markdown o JSON, el objetivo principal de la tabla es facilitar la comprensión humana de datos ordenados.

Características:

- Representación visual en filas y columnas, común en documentos, informes y bases de datos relacionales.
- Útil para visualización rápida y comparación entre valores.

Nombre	Edad	Ciudad
Ana	30	Madrid

- **XML (eXtensible Markup Language):**

XML (eXtensible Markup Language) es un lenguaje de marcado diseñado para almacenar, transportar y estructurar datos. Es ampliamente utilizado cuando se requiere una estructura jerárquica, legible tanto por humanos como por máquinas, especialmente en sistemas heredados y servicios web antiguos.

Características:

- Lenguaje basado en etiquetas para representar datos jerárquicos.
- Usado en configuraciones, servicios web y almacenamiento estructurado.

```
<persona>
  <nombre>Ana</nombre>
  <edad>30</edad>
  <ciudad>Madrid</ciudad>
</persona>
```


1.4. ¿Por qué estructurar?

Porque permite que las respuestas generadas por el modelo sean predecibles, reutilizables y procesables automáticamente. Una salida estructurada (como JSON, tablas u objetos) facilita la integración con sistemas externos, bases de datos, APIs o interfaces gráficas. Además, mejora la precisión en tareas específicas, como extracción de información, generación de reportes, automatización de flujos y validación de respuestas.

1.5. Salida estructurada con Modelos de lenguaje

Los LLMs son herramientas potentes capaces de generar texto coherente, traducir, resumir, clasificar y responder preguntas complejas. Sin embargo, en aplicaciones reales, no basta con que la salida sea solo comprensible para humanos: también debe ser estructurada y útil para otros sistemas.

¿Qué se espera de la salida ideal de un LLM?

Una salida útil y robusta debe cumplir con las siguientes características:

- Estructurada: organizada en un formato lógico, como JSON, tablas o listas jerárquicas.
- Legible para humanos y máquinas: fácilmente entendible por personas, pero también procesable automáticamente por software.
- Fácil de integrar: compatible con sistemas existentes como APIs, bases de datos, hojas de cálculo, dashboards, sistemas de recomendación o flujos de trabajo automatizados.

¿Por qué JSON es el formato más adecuado para salidas estructuradas en LLM?

JSON (JavaScript Object Notation) se ha consolidado como el formato estándar para la estructuración de datos generados por modelos de lenguaje, por varias razones clave:

1. Interoperabilidad

JSON es un formato universalmente aceptado en desarrollo de software, compatible con prácticamente todos los lenguajes de programación y plataformas. Esto permite conectar fácilmente la salida del modelo con cualquier otra herramienta o sistema (como APIs REST, bases de datos NoSQL, entornos de análisis de datos, etc.).

2. Legibilidad y simplicidad

A diferencia de XML u otros formatos más verbosos, JSON es ligero, limpio y fácil de leer, tanto para humanos como para sistemas. Esto reduce errores y mejora la experiencia de desarrollo.

3. Estructura clara

JSON permite definir estructuras jerárquicas, listas y objetos anidados, lo que representa bien la complejidad de muchos datos generados o extraídos por un LLM, como entidades con atributos, relaciones, o resultados por categorías.

4. Integración directa con frameworks de ML

Muchos frameworks de inteligencia artificial, aprendizaje automático y procesamiento de datos (como TensorFlow, PyTorch, Hugging Face, LangChain, etc.) aceptan y procesan datos en JSON de forma nativa, facilitando los flujos de desarrollo de aplicaciones.

5. Ideal para aplicaciones multimodales

En contextos multimodales (texto + imagen + audio), JSON permite agrupar y organizar distintas fuentes de entrada/salida de forma coherente.

2. Salida estructurada (JSON) y control de formato

En el contexto de los modelos de lenguaje, la salida en formato JSON (JSON Output) representa una evolución clave hacia la estructuración de respuestas generadas por inteligencia artificial. A diferencia del texto libre, este formato permite que la información sea organizada, procesada y reutilizada automáticamente por otros sistemas. Para asegurar que el JSON sea válido y funcional, se puede aplicar el parámetro `strip`, que al estar en `true` elimina espacios y caracteres innecesarios para producir una salida compacta, mientras que `strip = false` conserva una presentación más legible para humanos. Este tipo de control sobre el formato de salida es esencial cuando se integran modelos en sistemas automatizados o empresariales. Funcionalidades como `function calling` permiten que el modelo genere estructuras directamente invocables por software, mientras que los `outlines` ofrecen esquemas jerárquicos que descomponen el contenido en componentes clave. Comprender y aplicar estas herramientas permite generar distintos tipos de salidas (estructuradas, semiestructuradas o no estructuradas) según el objetivo, asegurando una interacción más precisa, confiable y útil entre el modelo y su entorno de aplicación.

2.1. ¿Qué es JSON Output?

JSON Output (salida en formato JSON) en el contexto de modelos de lenguaje o herramientas de generación de texto, nos referimos a la capacidad de obtener las respuestas generadas por el modelo directamente en un formato estructurado y estandarizado llamado JSON (JavaScript Object Notation). En lugar de producir texto libre y narrativo, el modelo genera su salida como un objeto de datos con clave y valor, organizado de manera jerárquica, clara y legible tanto para humanos como para máquinas.

Esto representa una evolución significativa en el uso de modelos de lenguaje, ya que permite que los resultados puedan ser automatizados, reutilizados y procesados directamente por aplicaciones, APIs o scripts, sin necesidad de interpretaciones adicionales o procesamiento de lenguaje natural posterior.

La salida en JSON resulta ideal en muchos escenarios prácticos donde se requiere:

- Automatizar la extracción de datos desde documentos, textos o formularios conversacionales.
- Rellenar estructuras de datos en formularios web, bases de datos o aplicaciones empresariales.
- Integrar la salida del modelo con APIs REST, dashboards, hojas de cálculo o sistemas inteligentes.
- Estandarizar respuestas para que múltiples herramientas puedan interpretarlas de la misma manera.
- Reducir errores de parsing o ambigüedad que pueden surgir en texto narrativo.

2.1.1. strip = true

Cuando se aplica Strip en la generación de salidas JSON a partir de un modelo de lenguaje, el resultado se procesa para eliminar cualquier carácter adicional que no forme parte estrictamente de la estructura JSON válida. Esto incluye espacios en blanco innecesarios, saltos de línea, tabulaciones o cualquier texto extra que pueda rodear o interrumpir el JSON. El objetivo es producir un JSON compacto y limpio, que generalmente aparece en una sola línea, sin elementos superfluos que puedan complicar su interpretación automática.

Este enfoque es especialmente útil en sistemas donde la salida JSON debe ser consumida por otros procesos, APIs o aplicaciones que esperan datos bien formateados y sin ruido. Además, ayuda a prevenir errores de parsing o validación que pueden ocurrir si la estructura contiene caracteres inesperados. La versión "limpia" facilita la integración, el almacenamiento eficiente y el procesamiento rápido, aunque puede ser menos legible para un humano en bruto debido a la ausencia de saltos de línea o indentación.

2.1.2. strip = false

Cuando no se aplica Strip en la salida, se genera un JSON de salida que puede incluir caracteres adicionales destinados a mejorar la legibilidad humana. Esto significa que la salida puede contener espacios en blanco, saltos de línea, tabulaciones y una estructura de sangrías –lo que se conoce como formato "pretty-print" o formato bonito– que hace más fácil entender visualmente la jerarquía y organización de los datos.

Aunque esta presentación mejora la claridad para quien lee directamente la salida, puede ser problemática para sistemas automáticos que requieren un JSON estrictamente válido y compacto. Sin embargo, en contextos donde la salida es revisada manualmente, depurada o documentada, mantener el formato con espacios y saltos de línea es muy beneficioso. Este modo favorece la inspección visual y la comprensión rápida, facilitando el desarrollo, debugging o análisis manual de la información generada.

2.2. Control del formato de salida

El **control del formato de salida** es un aspecto crucial en el uso de modelos de lenguaje para asegurar que las respuestas que generan no solo sean coherentes y contextuales, sino que también cumplan con requisitos estructurales específicos. Este control implica aplicar técnicas y herramientas que guían al modelo para producir resultados en formatos predefinidos y estructurados, como JSON, XML, tablas o esquemas jerárquicos, en lugar de texto libre o narrativo.

Este nivel de control es fundamental cuando los modelos se integran en sistemas complejos que dependen de datos precisos y organizados para funcionar correctamente. Por ejemplo, en aplicaciones de extracción de información, chatbots empresariales, interfaces de usuario inteligentes o automatización de flujos de trabajo, las respuestas deben ser interpretables por máquinas sin ambigüedad. Para lograrlo, se utilizan estrategias como la ingeniería de prompts específicos, funcionalidades nativas del modelo (como function calling), o frameworks y librerías externas que validan y formatean la salida. De esta manera, el control del formato permite convertir la capacidad generativa del modelo en una herramienta fiable y adaptable a escenarios técnicos diversos.

2.2.1. Function calling

La técnica de **function calling** es una funcionalidad avanzada presente en algunos modelos de lenguaje modernos que permite que el modelo no solo genere texto, sino que también “llame” o invoque funciones definidas externamente mediante la producción de una estructura de datos que representa los parámetros necesarios para ejecutar esa función. En lugar de entregar una respuesta en

lenguaje natural, el modelo genera un objeto de datos, típicamente en formato JSON, que contiene la información necesaria para que una función en el sistema receptor pueda ser llamada automáticamente con esos argumentos.

Esta funcionalidad permite una integración directa y robusta entre los modelos de lenguaje y los sistemas de software, facilitando flujos de trabajo donde la inteligencia artificial interactúa con bases de datos, APIs, servicios externos o lógica de negocio, sin necesidad de intermediarios que interpreten texto libre. Por ejemplo, en un asistente virtual que debe realizar una reserva o consultar información en tiempo real, el modelo puede devolver directamente los parámetros correctos para ejecutar esas acciones, garantizando precisión y eficiencia. Además, function calling mejora la predictibilidad y validación de las salidas, ya que las estructuras generadas deben ajustarse a los esquemas esperados por las funciones.

2.2.2. Outlines

Outlines son una representación estructurada, jerárquica y semántica de la información contenida en un texto. En lugar de presentar el contenido de forma lineal o narrativa, los outlines lo reorganizan en secciones bien definidas, categorías temáticas o roles informativos (como “quién”, “qué”, “cuándo”, “dónde”, “por qué” y “cómo”). Este enfoque permite descomponer textos complejos en partes comprensibles, resaltar los elementos clave y facilitar su análisis, resumen o reutilización posterior.

En el contexto de los modelos de lenguaje, generar un outline significa transformar una salida textual en una estructura lógica y predecible, que puede ser fácilmente visualizada, entendida y también procesada automáticamente. A menudo se utilizan outlines para:

- Resumir documentos largos o técnicos dividiéndolos en encabezados temáticos.
- Preparar contenido para presentaciones, dashboards o informes.
- Estandarizar respuestas en flujos de conversación o extracción de información.
- Transformar texto en esquemas reutilizables, por ejemplo, para guiones, formularios o bases de conocimiento.

2.3. Salida estructurada, semiestructurada y no estructurada

La salida estructurada, semiestructurada y no estructurada describe los distintos niveles de organización y formalidad que puede tener la información

generada por un modelo de lenguaje, y cada tipo tiene implicaciones diferentes para su procesamiento y utilidad.

- **Salida estructurada:** Se refiere a datos completamente organizados bajo un esquema claro y definido, con elementos claramente etiquetados y con relaciones explícitas. Ejemplos típicos son formatos como JSON, XML, CSV o tablas donde cada dato tiene un lugar preciso y un significado inequívoco. Este tipo de salida es ideal para procesamiento automático, integración con bases de datos y APIs, análisis estadístico o generación de reportes precisos. La salida estructurada facilita la reutilización, validación y automatización sin necesidad de procesamiento adicional.
- **Salida semiestructurada:** En este caso, la información tiene cierto grado de organización interna, pero no sigue un esquema rígido o formal. La estructura puede manifestarse a través de formatos como Markdown, listas con viñetas, etiquetas parciales o formatos de texto enriquecido, donde la información está parcialmente segmentada pero aún requiere procesamiento o interpretación para extraer los datos relevantes. Este tipo de salida es común en correos electrónicos, documentos con formato flexible o respuestas que contienen datos mezclados con texto libre.
- **Salida no estructurada:** Es el formato más libre y común en lenguaje natural, donde la información se presenta en texto continuo, sin etiquetas, ni organización formal. Aunque es natural y fácilmente entendible por humanos, su procesamiento automático es más complejo y requiere técnicas avanzadas de procesamiento de lenguaje natural (NLP) para extraer entidades, relaciones o información específica. Ejemplos típicos incluyen artículos, narrativas, conversaciones o comentarios sin formato.

Entender estos tres tipos de salida es fundamental para diseñar aplicaciones que utilicen modelos de lenguaje, ya que cada nivel tiene diferentes requerimientos técnicos para su integración, análisis y utilización efectiva.

3. Texto -> JSON estructurado con prompts y Json schema

La extracción estructurada de información a partir de texto requiere identificar entidades clave, relaciones entre datos y guiar al modelo hacia un formato preciso. Herramientas como NER, esquemas JSON, configuraciones como `json_mode` y funciones personalizadas permiten controlar la salida generada. Además, comparar enfoques tradicionales como spaCy con modelos de lenguaje actuales ayuda a elegir la mejor estrategia para transformar texto en datos útiles.

3.1. Named Entity Recognition (NER)

Named Entity Recognition (NER), o reconocimiento de entidades nombradas, es una técnica fundamental en el procesamiento de lenguaje natural (NLP) que consiste en identificar y clasificar automáticamente entidades específicas dentro de un texto, como nombres de personas, organizaciones, ubicaciones, fechas, cantidades, entre otros. El objetivo de NER es extraer información estructurada a partir de texto libre, facilitando tareas como el análisis de documentos, extracción de datos relevantes o automatización de formularios.

Por ejemplo, a partir del texto: “Compré unos Nike Air Force 1 en la sucursal de Monterrey el 12 de junio con mi tarjeta Banorte.”

Con NER aplicado, el modelo extrae e identifica automáticamente:

```
{
  "cliente": "anónimo",
  "producto": "Nike Air Force 1",
  "marca": "Nike",
  "sucursal": "Monterrey",
  "fecha": "12 de junio",
  "método_de_pago": "tarjeta Banorte"
}
```

NER puede utilizarse como etapa previa a convertir texto en estructuras como JSON, organizando las entidades clave en categorías útiles para el análisis o la integración con otros sistemas.

3.2. Extracción de relaciones

La extracción de relaciones va un paso más allá del NER: no solo identifica entidades, sino que también detecta los vínculos o relaciones que existen entre ellas dentro de un texto. Esta tarea busca construir estructuras como tripletas del tipo sujeto - relación - objeto, lo cual es clave en la creación de gráficos de conocimiento, motores de búsqueda inteligente o sistemas de respuesta automática.

Por ejemplo, en la frase: “*Laura Sánchez devolvió unos Adidas Ultraboost porque no le ajustaban bien.*”, se puede extraer la relación:

```
{
  "cliente": "Laura Sánchez",
  "acción": "devolvió",
  "producto": "Adidas Ultraboost",
  "motivo": "no le ajustaban bien"
}
```

Estas relaciones pueden representarse fácilmente en JSON y son esenciales para que un modelo comprenda conexiones semánticas complejas entre entidades dentro de un contexto.

3.3. Prompting para control estructural

El prompting para control estructural consiste en diseñar cuidadosamente los mensajes (prompts) que se dan a un modelo de lenguaje para obtener respuestas organizadas y predecibles en formatos como JSON, listas, tablas o esquemas jerárquicos. En lugar de pedir una respuesta libre, se le indica al modelo exactamente qué formato seguir, incluyendo campos específicos, nombres de claves, tipos de datos, o estructuras anidadas.

3.4. JSON API

Una JSON API es un estándar para enviar y recibir datos estructurados entre clientes y servidores, utilizando el formato JSON (JavaScript Object Notation). En el contexto de los LLMs, definir una JSON API implica establecer cómo debe estructurarse la salida del modelo para que pueda ser procesada directamente por sistemas informáticos. Al utilizar una JSON API, se especifica un esquema con atributos, tipos de datos y estructuras esperadas, permitiendo una comunicación clara, automatizada y sin ambigüedad.

Esto es especialmente útil cuando el modelo se utiliza como parte de un servicio más amplio, como rellenar formularios, generar objetos de datos o controlar acciones en una aplicación.

3.4.1. strict

strict asegura que la salida del modelo cumpla estrictamente con el esquema JSON definido, sin incluir elementos adicionales o formatos alternativos. Cuando `strict = true`, cualquier desviación del esquema (como campos extra o errores de sintaxis) es rechazada o corregida. Esto es ideal para garantizar integridad y confiabilidad en sistemas que consumen la salida del modelo sin intervención humana.

3.4.2. response_format=json

Esta opción indica explícitamente que se espera que la respuesta del modelo esté en formato JSON válido. Al utilizar `response_format=json`, se activa un modo de generación estructurada, evitando texto libre o narrativo. Esto



simplifica el parseo y reduce errores en flujos automáticos, especialmente cuando la salida se conecta directamente a código o servicios.

3.4.3. functions

functions permite definir funciones que el modelo puede “llamar” automáticamente, generando los parámetros requeridos en un formato estructurado (como JSON). En vez de generar texto, el modelo entrega un objeto que representa una llamada a función con argumentos listos para ejecutarse. Esto convierte al LLM en un intermediario inteligente que puede activar operaciones reales, como búsquedas, reservas o cálculos, desde lenguaje natural.

3.4.4. json_mode

`Json_mode` se refiere a activar o forzar al modelo para que toda su salida sea parsable como JSON, sin texto adicional o errores de formato que impedirían su lectura por un programa.

Hay dos maneras principales de lograr el `json_mode`:

- Mediante un system prompt claro y explícito.
- Mediante el parámetro `response_format`.

3.4.5. schema

Un `schema` es una especificación formal que describe la estructura, el formato y las restricciones que deben cumplir los datos representados en formato JSON. Su función principal es definir un conjunto de reglas claras y estrictas para validar que los documentos JSON tengan la forma correcta, incluyendo qué campos son obligatorios, qué tipos de datos se esperan (cadenas, números, listas, objetos), valores permitidos, patrones de formato y relaciones jerárquicas entre los elementos.

En el contexto de sistemas que generan o consumen datos automáticamente, como los modelos de lenguaje (LLMs), un `schema` garantiza que la información producida sea consistente, predecible y compatible con las aplicaciones que la van a procesar, evitando errores de interpretación, pérdida de datos o incompatibilidades entre sistemas. Además, facilita la integración entre diferentes componentes tecnológicos, como APIs, bases de datos, sistemas de validación y visualización, al proporcionar un contrato común que todas las partes entienden y respetan.

3.5. spaCy vs LLMs

spaCy es una biblioteca especializada en procesamiento de lenguaje natural tradicional (NLP), centrada en tareas como NER, análisis sintáctico, segmentación y tokenización. Es extremadamente rápida, ligera, y altamente precisa en dominios específicos, pero suele requerir entrenamiento adicional para tareas personalizadas. Por otro lado, los LLMs (Large Language Models) como GPT o Gemini son más versátiles y pueden adaptarse a tareas más amplias mediante prompting, sin necesidad de reentrenamiento, aunque a veces pueden ser menos consistentes o tener salidas ruidosas.

En resumen:

- spaCy es ideal para tareas bien definidas, de alto rendimiento y estructuras predecibles.
- LLMs son más flexibles, especialmente útiles cuando los datos son ambiguos, multimodales o se requiere generación de contenido más allá del análisis básico.

Una solución moderna puede integrar ambos: usar spaCy para tareas de extracción rápidas y controladas, y LLMs para estructuración compleja o contextos donde se requiera comprensión más profunda.

4. Imagen -> Estructura

En muchas aplicaciones del mundo real, la información no siempre está disponible en texto digital: se encuentra impresa en documentos físicos, incrustada en imágenes, o distribuida en archivos escaneados como tickets, facturas, formularios o reportes visuales. Para aprovechar esa información de forma automática y estructurada, es necesario convertir primero la imagen en texto legible, y luego organizar ese texto en formatos útiles como tablas o JSON. Este proceso, que va de imagen a estructura, se logra combinando técnicas de OCR (Reconocimiento Óptico de Caracteres) con modelos de lenguaje natural (NLP) o LLMs avanzados.

Este módulo explora cómo convertir imágenes en datos estructurados aprovechando tecnologías como OCR tradicional, modelos multimodales como GPT-4 Vision, y herramientas híbridas que combinan visión computacional con procesamiento de lenguaje. Desde la extracción de texto crudo hasta su interpretación semántica, aprenderemos cómo transformar capturas visuales en objetos de datos listos para ser usados en dashboards, bases de datos, reportes financieros o automatización empresarial.

4.1. ¿Qué es OCR?

El OCR es una tecnología que convierte texto de imágenes, como escaneados de documentos o imágenes de texto, en un formato digital editable. Esta tecnología permite extraer y reconocer el texto de una imagen y convertirlo en un formato que las máquinas pueden leer y manipular, como un archivo de texto editable.

Podemos identificar una línea de texto buscando líneas de píxeles blancos con líneas de píxeles negros en el medio.

Historia del OCR

Primeras ideas (1870-1931):

Los primeros conceptos de reconocimiento de texto surgieron con la invención de dispositivos como el optófono, desarrollado por Edmund Fournier d'Albe en 1913. Este aparato convertía patrones visuales en tonos sonoros, permitiendo a las personas ciegas "leer" letras impresas a través del oído. Aunque rudimentario, este invento marcó un hito al plantear la posibilidad de traducir información visual en señales comprensibles por otros sentidos, sentando las bases para el desarrollo del reconocimiento óptico de caracteres (OCR, por sus siglas en inglés).

Primeros inventos funcionales (1931-1954):

Durante este periodo se desarrollaron los primeros dispositivos capaces de interpretar información escrita de forma automatizada. En 1931, Gustav Tauschek patentó una máquina lectora de caracteres en Alemania, mientras que en los años 40 y 50 comenzaron a aparecer sistemas capaces de interpretar señales como el código Morse y convertirlas en salida de voz. Estos avances marcaron el inicio de la comercialización de tecnologías de lectura mecánica, especialmente con aplicaciones militares y de telecomunicaciones.

El OCR se vuelve portátil (1954-1974):

La miniaturización de los dispositivos permitió que el OCR comenzara a salir del laboratorio. Uno de los avances más significativos fue el desarrollo del Optacon (Optical-to-Tactile Converter), inventado en los años 60 por John Linvill y desarrollado por Telesensory Systems en 1971. Este dispositivo portátil permitía a las personas ciegas leer texto impreso al convertir las letras en patrones táctiles que podían sentir con el dedo. Aunque su uso requería entrenamiento, representó un gran avance en accesibilidad y autonomía personal.

OCR en todos lados (1974-2000):

Con la popularización de las computadoras personales y los escáneres, el OCR se integró a entornos comerciales y gubernamentales. Supermercados empezaron a usar lectores de códigos de barras para automatizar los procesos de cobro, mientras que los gobiernos implementaron OCR para la lectura automatizada de documentos como pasaportes y formularios censales. Los sistemas se volvieron más rápidos y precisos, aunque todavía dependían de fuentes tipográficas específicas y presentaban limitaciones con textos manuscritos o dañados.

OCR al alcance de todos (2000-2016):

La llegada de internet y la computación en la nube democratizó el acceso a herramientas de OCR. Aplicaciones como Adobe Acrobat, Google Drive y servicios gratuitos como WebOCR permitieron a cualquier persona convertir documentos escaneados en texto editable desde un navegador. Esta etapa se caracterizó por la integración del OCR en herramientas ofimáticas y la expansión de su uso en ámbitos educativos, empresariales y personales, sin necesidad de conocimientos técnicos avanzados.

OCR en la actualidad

Con la digitalización masiva de documentos y el uso generalizado de cámaras y escáneres, el OCR moderno trabaja casi exclusivamente sobre imágenes digitales. Esto ha ampliado su alcance, permitiendo extraer texto no solo de documentos escaneados, sino también de fotografías, capturas de pantalla o incluso transmisiones en video. Pero para entender cómo el OCR interpreta esta información, es fundamental comprender qué es exactamente una imagen digital, cómo se representa internamente en un sistema informático y de qué manera esos datos visuales pueden convertirse en texto legible para las máquinas.

Imagen Digital

Una imagen digital es una representación numérica de una imagen visual. Está compuesta por una matriz de elementos llamados píxeles, donde cada píxel contiene información sobre el color y la intensidad luminosa en una ubicación específica de la imagen.

Estas imágenes pueden obtenerse de diversas fuentes, como:

- Escáneres (documentos físicos)
- Cámaras digitales (fotografías)
- Capturas de pantalla
- Rayos X o imágenes médicas
- Satélites (imágenes remotas)

En el contexto del OCR, la imagen digital representa el “papel” sobre el cual se encuentran los caracteres que se desean reconocer.

Pixel

Un píxel es un punto único en una cuadrícula bidimensional que compone una imagen. Cada píxel tiene una posición específica y está asociado con un valor que determina su color y brillo. Estos valores pueden representarse en diferentes sistemas de color, como RGB (rojo, verde, azul) o CMYK (cian, magenta, amarillo, negro), dependiendo del dispositivo o el contexto.

A cada píxel se le asigna un valor tonal (negro, blanco, matices de gris o color), el cual está representado en un código binario (ceros y unos). Los dígitos binarios ("bits") para cada píxel son almacenados por una computadora en una secuencia, y con frecuencia se los reduce a una representación matemática (comprimida). Luego la computadora interpreta y lee los bits para producir una versión analógica para su visualización o impresión.

Densidad de píxeles

Densidad de píxeles es la cantidad de píxeles que hay en una imagen por cada pulgada, y se mide en PPI (píxeles por pulgada). Cuanto más alta sea la densidad, más pequeños y juntos están los píxeles, lo que hace que la imagen se vea más nítida y detallada. Si la densidad es baja, los píxeles son más grandes y visibles, y la imagen se ve menos definida.

Escala de grises

Una imagen en escala de grises es una imagen que no tiene colores, solo tonos que van desde el negro hasta el blanco, pasando por distintos grises.

- El negro representa la intensidad más baja (valor 0).
- El blanco representa la intensidad más alta (valor 255 en una escala de 8 bits).
- Los grises intermedios son distintos niveles de intensidad entre esos dos extremos.

Cada píxel en una imagen en escala de grises tiene un único valor numérico que indica su nivel de brillo, lo que la hace más simple que una imagen a color, donde se necesitan tres valores (como en RGB) por píxel.

Imagen Binaria

Una imagen binaria es una imagen digital con solo dos valores posibles para cada píxel.

Si bien las imágenes binarias pueden carecer del intrincado detalle de las imágenes en escala de grises o a color, son suficientes para aislar objetos y formas dentro de una imagen y se utilizan ampliamente en tareas de visión artificial.

Etapas del OCR

El proceso de OCR no se limita a simplemente "leer" el texto de una imagen. Implica una serie de etapas complejas y encadenadas que permiten preparar, extraer y reconocer los caracteres. Estas etapas se realizan mediante algoritmos específicos que trabajan sobre la imagen digital.

Las principales fases son: binarización, segmentación, adelgazamiento y comparación con patrones.

Binarización

En esta etapa, la imagen original (que puede estar en color o en escala de grises) se transforma en una imagen binaria, es decir, solo en blanco y negro. Esto se hace para diferenciar el texto del fondo de forma clara. Se aplica un valor de referencia, llamado umbral, para decidir qué píxeles se vuelven negros (texto) y cuáles blancos (fondo).

- Si la intensidad del píxel es mayor al umbral, se convierte en blanco (fondo).
- Si es menor, se convierte en negro (texto u objeto).

Esta conversión es importante porque simplifica mucho el procesamiento posterior, permitiendo identificar mejor las formas de las letras.

Segmentación

Una vez que la imagen está en blanco y negro, el siguiente paso es separar las diferentes partes del texto. Primero se detectan las líneas, luego las palabras dentro de esas líneas, y finalmente cada carácter. Esta división se hace analizando la distribución de los píxeles en la imagen. Es una etapa clave, porque si los caracteres no se aíslan correctamente, será muy difícil reconocerlos con precisión más adelante.

Adelgazamiento (Thinning)

Después de segmentar los caracteres, se realiza un proceso para simplificar su forma. Esto se llama adelgazamiento, y consiste en reducir el grosor de las letras, dejando solo una especie de “esqueleto” de cada una. Así, se eliminan detalles innecesarios, pero se conserva la estructura básica que define a cada letra. Esto hace más fácil compararlas con modelos de referencia durante el reconocimiento.

Comparación con patrones

En la última etapa, cada carácter extraído se compara con un conjunto de ejemplos ya conocidos para identificar qué letra, número o símbolo es. Esta comparación puede hacerse de forma directa (como si fuera una plantilla) o utilizando modelos avanzados, como redes neuronales, que han aprendido a reconocer letras a partir de miles de ejemplos. A veces, se tiene en cuenta también el contexto de las palabras para corregir posibles errores y mejorar la precisión.

4.2. ¿Cómo combinar OCR con NLP?

Combinar OCR (Reconocimiento Óptico de Caracteres) con NLP (Procesamiento de Lenguaje Natural) permite transformar información visual (como imágenes de tickets, facturas, formularios o documentos escaneados) en texto estructurado y analizable automáticamente. Esta combinación es clave para automatizar flujos de trabajo donde los datos importantes se encuentran en formato impreso o manuscrito, y deben ser comprendidos por sistemas inteligentes.

4.3. GPT-4 Vision: ¿Cómo analiza imágenes y estructuras contenido?

GPT-4 Vision es una extensión del modelo GPT-4 que permite analizar y razonar sobre contenido visual, como imágenes, fotografías, diagramas, gráficos o tablas. A diferencia de modelos que solo procesan texto, GPT-4 Vision combina texto e imagen para generar respuestas más ricas, precisas y útiles.

Este tipo de modelo multimodal tiene la capacidad de identificar objetos, leer texto dentro de imágenes, interpretar relaciones visuales y convertir esa información en formatos estructurados como listas, tablas o JSON.

¿Cómo analiza imágenes GPT-4 Vision?

GPT-4 Vision convierte la imagen en representaciones internas que combinan características visuales (formas, colores, texto, disposición) con su comprensión lingüística del mundo. Esto le permite:

- Leer y transcribir texto presente en una imagen (como si hiciera OCR).
- Entender estructuras como cuadros, tablas o formularios.
- Interpretar jerarquías visuales (por ejemplo, títulos, columnas o agrupaciones).
- Generar contenido textual basado en lo que “ve”.

Al recibir una instrucción como “extrae la tabla en formato JSON”, GPT-4 analiza visualmente la imagen, identifica los elementos clave y organiza los datos conforme a la estructura solicitada.

Descripción automática de imágenes

GPT-4 Vision también es capaz de generar **descripciones detalladas de imágenes**, interpretando tanto su contenido visual como su significado contextual. Esta función es especialmente útil para generar texto accesible para personas con discapacidad visual, crear metadatos automáticos o resumir el contenido de una imagen.

Ejemplos de uso incluyen:

- Describir lo que ocurre en una fotografía (personas, acciones, expresiones).
- Generar pies de foto automáticos.
- Detectar emociones, relaciones espaciales o detalles específicos.

Un prompt típico sería: "Describe en detalle lo que aparece en esta imagen" o "¿Qué está ocurriendo en esta escena?". El modelo responderá con una interpretación basada en su comprensión general del mundo y del contenido visual.

Ejemplo:

Prompt: *Dame una descripción detallada de la imagen. Cuantas personas, objetos u elementos hay en la imagen y que actividad se está realizando?*



Respuesta: *En la imagen hay cuatro personas: dos adultos y dos niños. Están al aire libre, aparentemente en una playa o cerca de un cuerpo de agua, al atardecer o amanecer, ya que el cielo tiene tonos cálidos y el sol está cerca del horizonte.*

Los adultos están en los extremos, uno a la izquierda y otro a la derecha, y los dos niños están en el centro. El adulto de la derecha parece estar lanzando una pelota hacia los niños, quienes están levantando las manos para atraparla.

La actividad que se está realizando es un juego familiar en el que están jugando a lanzarse una pelota.

En cuanto a objetos, el único visible es la pelota que está en el aire entre los adultos y los niños. También hay hierba o plantas bajas en el suelo alrededor de ellos. La escena transmite un momento de diversión y convivencia en familia al aire libre, con un fondo natural muy llamativo por la puesta de sol.

Identificación de lugares en imágenes

Una capacidad destacada de GPT-4 Vision es su habilidad para identificar lugares, escenarios y ubicaciones geográficas dentro de imágenes. El modelo puede analizar paisajes, monumentos, arquitectura y señales visibles en la escena para inferir de qué lugar se trata. Aunque no es un sistema de geolocalización exacta, puede proporcionar descripciones útiles como "una playa tropical", "una ciudad europea con arquitectura gótica" o "un edificio moderno posiblemente en Asia oriental".

Esto lo hace útil en aplicaciones como:

- Clasificación automática de fotos por ubicación general.
- Análisis de imágenes turísticas o culturales.
- Soporte para descripciones accesibles de fotografías.

La calidad de esta identificación depende de la claridad visual, la presencia de elementos distintivos y el contexto proporcionado en el prompt.

Prompt: *Identifica la ciudad de la imagen*



Respuesta: *La imagen muestra una vista de Mazatlán, una ciudad ubicada en la costa del Pacífico en el estado de Sinaloa, México. El edificio destacable es el "Castillo del Crematorio" o "El Fuerte", un lugar icónico de la zona.*

4.4. OCR con LLMs

OCR con LLMs (Reconocimiento Óptico de Caracteres combinado con Modelos de Lenguaje de gran escala) es una técnica avanzada que une dos etapas clave: la extracción de texto desde imágenes mediante OCR, y la interpretación, estructuración o enriquecimiento de ese texto utilizando un modelo de lenguaje (como GPT, Gemini o Claude). Este enfoque potencia la transformación de documentos visuales –como tickets, recibos, facturas, contratos o formularios escaneados– en salidas estructuradas, legibles y útiles para sistemas automatizados.

En la primera etapa, un motor OCR convierte píxeles en texto plano. Sin embargo, ese texto suele carecer de estructura semántica o contexto: puede tener errores, líneas desordenadas o información difícil de segmentar. Aquí es donde entran los LLMs: interpretan el texto extraído y lo reorganizan en una forma estructurada, aplicando comprensión contextual, detección de entidades, inferencias y formateo.

4.5. Convertir imagen en JSON: extracción tabular

Una de las aplicaciones más potentes es convertir información visual (como una tabla en una imagen) en un formato estructurado como JSON, útil para sistemas, bases de datos o análisis automatizado.

Además de tablas, GPT-4 Vision puede extraer:

- Listas ordenadas o desordenadas.
- Campos tipo formulario (nombre, fecha, monto, firma).
- Jerarquías visuales (títulos, subtítulos).
- Pares clave-valor (comunes en recibos, facturas, reportes).

También es posible generar una salida estructurada mediante la identificación de objetos. Por ejemplo, si se le proporciona una imagen con varios productos o vehículos, el modelo puede contar cuántos hay, clasificarlos por tipo, y generar una lista o estructura en formato JSON que representa los objetos detectados junto con sus características (tipo, color, posición, cantidad, etc.). Esto permite estructurar información visual sin que necesariamente haya texto en la imagen.

Todo esto depende del prompt que se le proporcione, lo que le permite ser flexible según la necesidad del usuario.

4.5.1. Salida estructurada desde imagen

La salida estructurada desde imagen se refiere al proceso mediante el cual se transforma información visual contenida en una imagen (como un ticket, factura, tabla impresa, recibo o documento escaneado) en un formato organizado, jerárquico y procesable automáticamente, como JSON, CSV o tablas estructuradas. Este enfoque combina tecnologías de visión computacional con herramientas de procesamiento de lenguaje natural (NLP) o modelos de lenguaje (LLMs) para interpretar y dar sentido al texto extraído.

En lugar de simplemente devolver un bloque plano de texto, el sistema analiza la semántica, posición y contexto del contenido visual para asignar etiquetas y construir una estructura coherente. Por ejemplo, al procesar una imagen de un ticket de compra, se identifican y organizan campos como el nombre del cliente, productos adquiridos, totales, fecha y sucursal, devolviendo un JSON estructurado que puede integrarse directamente en bases de datos, reportes o aplicaciones empresariales.

Este tipo de salida estructurada permite automatizar tareas que antes requerían revisión humana, como digitalización de registros físicos, validación contable, captura masiva de datos o análisis de formularios escaneados, haciendo posible extraer valor real y utilizable desde imágenes sin necesidad de interpretación manual.

4.5.2. Salida estructurada con modelos no generativos

La salida estructurada con modelos no generativos se refiere al proceso de transformar texto o datos visuales (como los extraídos por OCR) en formatos organizados y fácilmente procesables –como JSON, tablas o CSV– utilizando herramientas tradicionales de procesamiento de lenguaje natural (NLP) basadas en reglas, árboles de decisión o modelos estadísticos, sin depender de modelos generativos grandes como GPT o Gemini.

A diferencia de los LLMs, que pueden generar texto complejo y adaptativo, los modelos no generativos (como los basados en spaCy, NLTK o reglas regulares) funcionan con patrones explícitos o algoritmos entrenados para tareas específicas como:

- Reconocimiento de entidades (NER)
- Etiquetado POS (Part-of-Speech)
- Extracción de relaciones
- Parsing sintáctico
- Segmentación por layout

5. Comparación Multimodal: texto vs imagen

Cuando se procesan con modelos de inteligencia artificial, particularmente con modelos de lenguaje multimodales. Cada modalidad aporta distintos tipos de información y desafíos técnicos: el texto es explícito y estructurable; la imagen es visual y espacial; Esta sección compara sus ventajas, limitaciones y consideraciones al momento de estructurar datos provenientes de cada fuente.

5.1. Dificultades de cada modalidad

- **Texto:** Aunque es la modalidad más directa y estructurable, puede presentar ambigüedades semánticas, lenguaje informal o desorganizado (como chats o mensajes de voz transcritos). Además, el texto puede contener errores ortográficos, formatos inconsistentes y falta de delimitadores claros que dificultan su estructuración.
- **Imagen:** La principal dificultad es que los datos no están explícitamente accesibles; deben interpretarse visualmente. Las imágenes pueden tener ruido visual, iluminación deficiente, distorsión, texto curvo o manuscrito. Además, la segmentación de elementos (como celdas en una tabla o campos en un ticket) puede ser imprecisa, y el OCR no siempre garantiza resultados limpios.

5.2. Tipo de errores estructurales comunes

Texto:

- Campos incompletos o mal detectados
- Confusión entre entidades similares (ej. nombre vs. producto)
- JSON mal cerrado o desalineado con el esquema esperado

Imagen:

- OCR devuelve texto roto o desordenado
- Mala segmentación de campos visuales
- Pérdida de jerarquía visual (por ejemplo, subtítulos dentro de tablas)

5.3. ¿Qué pasa cuando no se puede controlar la salida?

Cuando no se puede controlar la salida de un modelo (por ejemplo, en modelos generativos sin restricciones o sin validación de formato), se corre el riesgo de obtener respuestas **inconsistentes, ambiguas o no procesables automáticamente**. En contextos empresariales o de automatización, esto puede resultar en:

- Fallos de parsing al intentar leer un JSON mal formado
- Errores en el flujo de datos hacia bases de datos o APIs
- Resultados que necesitan revisión manual, rompiendo la escalabilidad
- Pérdida de trazabilidad, confianza o cumplimiento normativo

Por eso es crucial implementar **mecanismos de validación**, como el uso de schema, salidas en `json_mode`, function calling, o frameworks como Pydantic. Estos métodos permiten forzar al modelo a seguir una estructura definida y asegurar que la salida sea útil y reutilizable por otros sistemas. Sin este control, la multimodalidad pierde confiabilidad como herramienta en pipelines automatizados.

6. Audio -> Texto -> Estructura con Whisper y Gemini

La conversión de audio en datos estructurados es clave para automatizar tareas como análisis de conversaciones, soporte al cliente o ventas por voz. Tecnologías como **Whisper**, un modelo de reconocimiento automático del habla (ASR), permiten transcribir audios con gran precisión. A partir de esa transcripción, modelos como **BERT** o **LLMs** pueden extraer información organizada en formatos como JSON.

Por su parte, **Gemini**, como modelo multimodal, permite trabajar directamente con audio (y otras modalidades) para generar estructuras útiles sin pasos intermedios. Este módulo explora cómo funcionan estas herramientas, sus variantes, modos de uso y cómo aplicarlas para convertir voz en datos listos para usar.

6.1. ASR (Automatic Speech Recognition)

El Reconocimiento Automático del Habla (ASR, por sus siglas en inglés) es la tecnología que permite convertir audio hablado en texto escrito. También se conoce como Speech-to-Text (STT). Esta tecnología se basa en modelos de inteligencia artificial entrenados con miles de horas de grabaciones de voz y sus transcripciones correspondientes. El objetivo principal del ASR es identificar fonemas, palabras y oraciones en una señal de audio, interpretarlas correctamente y generar una transcripción coherente en lenguaje natural.

Aunque el ASR tiene múltiples aplicaciones (como subtítulo automático, asistentes de voz, accesibilidad, etc.), en nuestro caso se utiliza como el primer paso en un flujo de extracción estructurada de datos desde audio. Esto significa que el audio no se transcribe simplemente para ser leído, sino que el texto resultante será analizado por un modelo de lenguaje para identificar

entidades clave, relaciones semánticas y estructuras útiles para crear datasets automáticos.

6.2. Whisper

Whisper es un sistema de transcripción automática de audio desarrollado por OpenAI, entrenado con grandes cantidades de datos multilingües y multiformato. Su función principal es convertir audio hablado (como entrevistas, grabaciones, videos o llamadas) en texto escrito. Whisper es especialmente eficaz en entornos donde la calidad del audio puede variar, y es capaz de manejar múltiples idiomas, acentos y condiciones acústicas. Se utiliza en tareas como subtítulos, transcripción de reuniones, accesibilidad, y como etapa inicial en flujos de extracción estructurada desde audio.

6.2.1. Variantes

Whisper viene en varias **versiones (o tamaños)** según la capacidad y velocidad necesarias:

- tiny, base, small: rápidos, ideales para tareas simples o en dispositivos limitados.
- medium, large: más lentos pero mucho más precisos.
- Whisper-v3: la versión más actualizada, con mejoras en multilingüismo y velocidad.

La elección de la variante depende del balance entre **velocidad, precisión y recursos computacionales**.

6.2.2. Chunks

Whisper no procesa archivos de audio enteros de una sola vez; en su lugar, los divide en "chunks" o fragmentos de unos segundos (por ejemplo, 30 segundos), que luego transcribe individualmente. Este enfoque mejora el rendimiento, pero puede fragmentar frases o romper contextos largos. Para solucionar esto, se recomienda el uso de postprocesamiento que una, corrija o recontextualice las partes transcritas.

6.2.3. Procesamiento con BERT

Una vez transcrito el texto, **modelos tipo BERT (Bidirectional Encoder Representations from Transformers)** pueden usarse para **analizar, clasificar o extraer entidades** del contenido transcrito. BERT es muy efectivo para tareas como:

- Extracción de intención (ej. compra, queja, consulta)
- Identificación de entidades nombradas (producto, lugar, fecha)
- Clasificación de sentimientos en llamadas

Este pipeline **Whisper → BERT** es común para tareas de análisis de voz en aplicaciones empresariales.

6.2.4. Procesamiento con LLM

Alternativamente, los textos transcritos por Whisper pueden ser enviados a un **LLM (modelo de lenguaje grande)** como GPT-4 o Gemini para una **extracción estructurada más compleja**. Esto permite:

- Extraer múltiples campos en formato JSON
- Identificar relaciones entre entidades (producto y precio)
- Generar resúmenes estructurados o insights automáticos

6.3. Gemini

Gemini es el modelo multimodal de Google que no solo comprende texto, sino también imágenes y audio. A diferencia de Whisper, Gemini puede combinar entradas visuales y auditivas para generar salidas más complejas y contextualizadas. También puede analizar audio directamente (sin necesidad de Whisper como intermediario), interpretar tono, extraer intenciones y entregar salidas estructuradas.

6.3.1. Extracción desde Audio

Gemini puede recibir archivos de audio y devolver:

- Transcripciones limpias
- Análisis semántico (intención, sentimiento)
- JSON estructurado con campos clave

6.3.2. Opciones

Gemini ofrece múltiples formas de trabajo:

- Solo transcripción (como Whisper)
- Transcripción + extracción estructurada en una sola llamada
- Análisis multimodal combinando texto, imagen y audio
- Llamadas en streaming o archivos largos

Esto lo hace ideal para integrar en sistemas que reciben datos complejos (audio + foto de ticket, por ejemplo) y requieren respuestas estandarizadas y contextualizadas.

6.3.3. Costos

- **Whisper** es **open-source** y **gratuito**, pero requiere GPU para alto rendimiento.
- **Gemini** es de uso comercial y puede tener un **costo por token procesado**, especialmente si se incluye audio o imagen.
- Para tareas frecuentes y ligeras, Whisper es más económico.
- Para flujos complejos, multimodales o que requieran razonamiento estructurado, Gemini justifica su costo.

7. Extras

En este último bloque se abordan elementos complementarios pero clave para llevar la estructuración de datos con modelos de lenguaje a entornos reales de desarrollo. Por un lado, herramientas como **JSON API** y **Pydantic** permiten validar y tipar estructuras de datos para asegurar su correcta integración. También exploramos **patrones de reintento**, útiles para gestionar errores o salidas no válidas durante la generación automática. Además, se presentan **otras modalidades** más allá de texto, imagen o audio, como video o sensores, y se revisan **otros proveedores** de modelos que ofrecen capacidades estructuradas, brindando un panorama más amplio de opciones en el ecosistema de IA.

7.1. JSON API y pydantic

JSON API es una especificación estándar que define cómo estructurar solicitudes y respuestas en formato JSON para servicios web. Su objetivo es proporcionar una forma coherente y predecible de intercambiar datos entre cliente y servidor,

minimizando ambigüedades y errores. En el contexto de modelos de lenguaje, usar JSON API permite que las salidas generadas por un modelo se integren fácilmente con otras herramientas o aplicaciones que siguen esta convención.

Por otro lado, Pydantic es una librería de validación de datos en Python basada en tipos (type hints). Permite definir modelos de datos como clases que validan automáticamente la estructura y tipos de un JSON. Cuando se usa con modelos de lenguaje, Pydantic actúa como un puente de validación, asegurando que el JSON generado por el modelo cumpla con los requisitos esperados antes de ser procesado por una aplicación o API. Esto es especialmente útil para evitar errores en producción y garantizar que la IA produzca salidas estructuradas, válidas y coherentes.

7.2. Patrones de reintento

Los **patrones de reintento** (retry patterns) son estrategias utilizadas para manejar errores o fallas temporales durante la ejecución de una tarea, como una llamada a un modelo de lenguaje o una API. En el contexto de salidas estructuradas, se aplican cuando el modelo genera una respuesta incompleta, inválida o que no se ajusta al esquema requerido (por ejemplo, un JSON mal formado).

Un patrón de reintento puede incluir:

- Repetir el prompt con ajustes menores.
- Aplicar validaciones (como con Pydantic) antes de aceptar la respuesta.
- Incrementar el número de reintentos con cada fallo, hasta un límite.
- Agregar instrucciones adicionales al modelo para reforzar el formato.

Estos patrones son fundamentales para garantizar la robustez de los sistemas que dependen de modelos generativos, especialmente en producción o automatización de flujos.

7.3. Otras modalidades

Además de texto, imagen y audio, existen **otras modalidades** que también pueden estructurarse y analizarse con IA multimodal. Algunas de las más relevantes incluyen:

- **Video:** que combina imagen + audio + movimiento; útil para análisis de seguridad, contenido educativo o seguimiento de comportamiento.



- **Datos sensoriales o espaciales:** como señales biométricas (ECG, EEG), sensores IoT, datos GPS o mapas.
- **Lenguaje de señas:** captado por video y convertido a texto mediante modelos especializados.

Estas modalidades expanden las capacidades de los modelos de lenguaje más allá de la conversación tradicional, permitiendo aplicaciones en salud, robótica, educación, industria y accesibilidad.

7.4. Otros proveedores

Aunque OpenAI, Google (Gemini) y Anthropic (Claude) dominan gran parte del mercado de modelos de lenguaje, existen **otros proveedores** con capacidades relevantes para tareas estructuradas o multimodales:

- **Cohere:** conocido por su enfoque en NLP estructurado y clasificación.
- **Mistral:** especializado en modelos abiertos de alto rendimiento.
- **Meta (con LLaMA):** ofrece modelos open-source con capacidades robustas.
- **Hugging Face:** provee acceso a miles de modelos preentrenados, incluyendo los de extracción estructurada.
- **AWS Bedrock y Azure OpenAI:** permiten integrar modelos comerciales en entornos empresariales con control, escalabilidad y seguridad.

La elección del proveedor depende del caso de uso, costos, restricciones de privacidad, disponibilidad en el entorno de trabajo y capacidades específicas (multilingüismo, acceso a APIs, personalización, etc.).



Bibliografía y Recursos Adicionales

1. Google AI Studio: <https://aistudio.google.com/>
2. Documentación del taller: <http://dptaller.certexai.com/>
3. Open AI: <https://platform.openai.com/>