ChatGPT

Integración de Document AI, Vertex AI Gemini 2.5 Flash y A2A en un Agente Multimodal

En este documento se detalla cómo aprovechar **Google Cloud Document AI**, los **modelos generativos Gemini 2.5 Flash** de Vertex AI (multimodales) y el **protocolo Agent-to-Agent (A2A)** para construir un agente de IA capaz de analizar imágenes catastrales (por ejemplo, PDFs con mapas o texto) y brindar respuestas fundamentadas. Se cubre qué ofrece cada servicio y cómo implementarlos en una solución Python, incluyendo la opción de *grounding* (búsqueda web) y la integración con un agente A2A.

Google Cloud Document AI: extracción de información de documentos

Document AI es la plataforma de Google Cloud para entender documentos. Permite convertir datos no estructurados de documentos (imágenes, PDFs, etc.) en datos estructurados listos para usar [1]. En otras palabras, Document AI emplea técnicas de procesamiento de lenguaje natural y visión por computador para extraer texto y estructurar información de documentos escaneados, formularios, imágenes con texto, etc. Esto incluye tareas como:

- **OCR (Reconocimiento Óptico de Caracteres)** sobre documentos (identifica texto impreso o manuscrito y su disposición en páginas) [1] [2]. Por ejemplo, puede leer texto de imágenes catastrales o PDFs escaneados, incluso en más de 200 idiomas [3].
- **Detección de estructura y campos**: identifica la estructura del documento (párrafos, bloques, tablas, casillas marcadas) y extrae pares clave-valor y entidades (fechas, nombres, números, etc.) si se trata de formularios o documentos estructurados [1] [2].
- **Procesadores especializados**: además del OCR genérico, Document AI ofrece *pre-trained models* para tipos de documento específicos (facturas, recibos, contratos, IDs, etc.) que extraen información relevante de cada tipo [4]. En nuestro caso (imágenes catastrales con texto), probablemente usaremos el **procesador general de OCR**, que es capaz de manejar documentos en diversos formatos y extraer todo el texto y la disposición/layout [3].

Implementación (uso de Document AI con Python): Para utilizar Document AI en nuestro agente, es necesario habilitar la API de Document AI en el proyecto de GCP y crear un *Processor* (procesador) de tipo OCR en la consola de Google Cloud [5]. Al crear el procesador obtendremos un `processor_id` único. Luego, podemos invocar la API desde Python mediante el cliente oficial. Pasos generales:

1. **Configuración de credenciales:** usar una cuenta de servicio de GCP con permisos en Document AI y establecer la variable de entorno `GOOGLE_APPLICATION_CREDENTIALS` apuntando al JSON de credenciales [6] [7].
2. **Instalar la librería cliente:** `pip install --upgrade google-cloud-documentai` [8].
3. **Código para procesar un documento:** utilizar el cliente `DocumentProcessorServiceClient` para enviar el archivo (imagen o PDF) al procesador OCR y obtener la respuesta. Por ejemplo, el siguiente fragmento ilustra cómo leer un PDF local y extraer su texto con Document AI [9] [10]:

```python
from google.cloud import documentai_v1 as documentai
from google.api_core.client_options import ClientOptions

# ID del proyecto, ubicación y ID del procesador OCR (obtenido de la consola)
project_id = "MI_PROYECTO_ID"
location = "us"  # e.g. "us" o "eu" según la región del procesador
processor_id = "XXXXXXXXXXXXXXXX"

# Configurar el endpoint según la ubicación
opts = ClientOptions(api_endpoint=f"{location}-documentai.googleapis.com")
client = documentai.DocumentProcessorServiceClient(client_options=opts)

# Construir el nombre completo del procesador
name = client.processor_path(project_id, location, processor_id)
print(f"Usando procesador: {name}")

# Leer el archivo PDF o imagen en binario
file_path = "path/al/archivo.pdf"
with open(file_path, "rb") as file:
    file_content = file.read()

# Crear el documento bruto con el contenido y tipo MIME
raw_document = documentai.RawDocument(content=file_content,
mime_type="application/pdf")

# Preparar y enviar la solicitud de procesamiento
request = documentai.ProcessRequest(name=name, raw_document=raw_document)
result = client.process_document(request=request)

# Extraer el texto del resultado
document = result.document
texto_extraido = document.text
print("Texto extraído:", texto_extraido)
```

En este ejemplo, tras enviar el documento al procesador OCR, obtenemos un objeto `Document` cuya propiedad `text` contiene todo el texto detectado en el PDF/imágenes [10]. Adicionalmente, el objeto `Document` incluye estructura del layout (páginas, coordenadas de bloques de texto, etc.), lo cual permite localizar dónde aparece cada fragmento en la imagen original si es necesario.

1. **Post-proceso de la información:** Una vez obtenido el texto y datos estructurados, el agente puede filtrar o interpretar la información relevante. Por ejemplo, para imágenes catastrales, podríamos buscar campos como número de parcela, dirección, superficies, etc., dentro del texto extraído.

   Nota: Document AI puede complementarse con **Cloud Vision API** en ciertos casos. Cloud Vision es otro servicio de Google Cloud orientado a análisis de imágenes (clasificación, detección de objetos, etc.). En este proyecto, ya se cuenta con un componente de *Vision AI* como "verificador" en el agente; por ejemplo, se podría usar Vision AI para validar que la

imagen de entrada sea del tipo correcto (un mapa catastral legible) antes de pasarla a Document AI, o para extraer rápidamente texto vía detección de texto simple. Sin embargo, Document AI OCR suele ser más robusto para extraer texto denso de documentos, por lo que será la pieza central para obtener la información textual de imágenes/PDFs.

Vertex AI Gemini 2.5 Flash: modelos generativos multimodales

Vertex AI Gemini es la familia de modelos de lenguaje de última generación de Google (de DeepMind) disponibles en Google Cloud. Los modelos **Gemini 2.5** se caracterizan por ser **multimodales de forma nativa**, es decir, pueden aceptar distintos tipos de entrada (texto, imágenes, audio e incluso video) y producir respuestas en texto [11]. Esto los hace ideales para casos donde necesitamos que la IA razone sobre texto junto con imágenes. En particular, **Gemini 2.5 Flash** es una variante optimizada para **alta velocidad y costo menor**, manteniendo buenas capacidades generales de razonamiento [12]. Algunas características destacadas de *Gemini 2.5 Flash*:

- *Equilibrio costo-rendimiento:* Es el modelo de mejor relación precio/rendimiento de la serie, adecuado para aplicaciones que requieren respuestas rápidas pero suficientemente inteligentes [12].
- *Capacidades de "pensamiento" explícito:* Es el primer modelo *Flash* que soporta **thinking capabilities**, lo que significa que puede exponer su proceso de razonamiento interno antes de dar la respuesta [12]. En la práctica, esto permite obtener explicaciones o trazas del pensamiento del modelo (útil para debugging o para mayor transparencia en entornos empresariales).
- *Contexto extenso:* Admite hasta **1 millón de tokens** de entrada (máximo) y respuestas de decenas de miles de tokens [13]. Esto implica que podemos proporcionarle documentos largos (incluso cientos de páginas) en la entrada, o combinar múltiples fuentes (texto de documentos + descripciones de imágenes) dentro de un mismo prompt.
- *Soporte multimodal:* Acepta **imágenes, audio, videos y código** además de texto como parte del prompt [11], integrándolos en su proceso de generación de respuesta. La salida siempre será texto (por ejemplo, una explicación o un resumen). En nuestro caso, podríamos alimentar al modelo con el texto extraído de Document AI *y* potencialmente la imagen misma o partes de ella, para que el modelo tenga el contexto completo de la imagen catastral.
- *Funciones avanzadas integradas:* Gemini 2.5 Flash puede utilizar herramientas adicionales durante la generación. Por ejemplo, soporta **llamadas a funciones** definidas, ejecución de código (vía herramientas de código), y ***grounding*** con fuentes externas [14]. Esto último incluye la posibilidad de consultar la web (Google Search) para obtener información actualizada y fundamentar mejor sus respuestas, como detallaremos en la siguiente sección.

Implementación (consumir Gemini 2.5 Flash en Vertex AI): Para usar este modelo, debemos tener habilitada la API de Vertex AI Generative AI. Hay varias formas de invocar a Gemini desde Python; una de las más sencillas es usar el **SDK Google GenAI** (`google-genai`), que unifica llamadas a los modelos generativos de Vertex. Pasos básicos:

1. **Instalar el SDK genai:** `pip install --upgrade google-genai` [15]. Este SDK facilita las llamadas a modelos como Gemini. Asegúrate de tener las credenciales adecuadas (p. ej., usar la misma cuenta de servicio o autenticación ADC configurada previamente).
2. **Configurar el entorno:** Debes especificar el proyecto de Google Cloud y la ubicación. Por ejemplo, establecer las variables de entorno `GOOGLE_CLOUD_PROJECT` (ID del proyecto) y `GOOGLE_CLOUD_LOCATION` (por ejemplo, "global") antes de usar el SDK, y activar el uso de Vertex

con `GOOGLE_GENAI_USE_VERTEXAI=True` [16]. Esto garantiza que las peticiones se dirijan a Vertex AI en tu proyecto.

3. **Llamar al modelo Gemini 2.5 Flash:** Utiliza el cliente del SDK para generar contenido. Puedes especificar el modelo por nombre (`"gemini-2.5-flash"`) y pasar el prompt en el parámetro `contents`. Opcionalmente, se pueden incluir **herramientas de grounding** (como búsqueda web) en la configuración. Un ejemplo de código en Python para hacer una pregunta al modelo con *grounding* web habilitado sería:

```
from google import genai
from google.genai.types import GenerateContentConfig, Tool, EnterpriseWebSearch

client = genai.Client()  # Usa credenciales ADC del entorno (proyecto ya
configurado)
prompt = "¿Cuál es la capital de Australia y cuántos habitantes tiene
aproximadamente?"
response = client.models.generate_content(
    model="gemini-2.5-flash",
    contents=prompt,
    config=GenerateContentConfig(
        tools=[ Tool(enterprise_web_search=EnterpriseWebSearch()) ]
    )
)
print(response.text)
```

En este ejemplo, `model="gemini-2.5-flash"` indica que usamos Gemini 2.5 Flash, y pasamos una lista de herramientas donde incluimos `EnterpriseWebSearch()` para que el modelo realice una búsqueda web si lo necesita [17]. La respuesta (`response.text`) será un texto generado por la IA que responde la pregunta, posiblemente citando o basándose en información actual de la web (gracias al *grounding*).

Si no se necesitara *grounding*, simplemente omitiríamos la parte de `tools` en la configuración, y el modelo respondería basado sólo en su conocimiento entrenado (hasta 2023) y el contexto proporcionado.

1. **Incluir contenido multimodal:** Si deseamos proporcionarle una **imagen o PDF** directamente al modelo Gemini (además o en lugar del texto), esto es posible ya que es multimodal. En la API de Vertex AI, podemos adjuntar archivos o datos binarios en el prompt. Por ejemplo, podríamos convertir la imagen catastral a texto base64 o subirla a Cloud Storage y referenciarla. Los modelos Gemini 2.5 admiten **hasta 3,000 imágenes o documentos** por consulta (con tamaños y páginas máximas) [18] [19], por lo que podemos incrustar la imagen completa. En muchos casos prácticos, sin embargo, es más eficiente usar Document AI para extraer el texto relevante de la imagen y pasar ese texto como contexto, en lugar de darle la imagen cruda al modelo generativo. Aun así, la capacidad está ahí: por ejemplo, podrías incluir en `contents` no sólo texto sino objetos de tipo `Image` o `Document` según el SDK, permitiendo que Gemini procese la información visual directamente. (Cabe señalar que el manejo detallado de adjuntar archivos binarios a la solicitud va más allá del alcance de este resumen, pero la documentación de Vertex AI provee guías para enviar imágenes como parte del prompt multimodal).

Grounding: respuestas fundamentadas con búsqueda web (*Web Grounding*)

Una potente funcionalidad de los modelos Gemini es su capacidad de **grounding**, es decir, complementar sus respuestas con información obtenida de fuentes externas en tiempo real. En particular, *grounding con búsqueda web* permite que el modelo realice consultas a un buscador (Google) para traer datos actualizados o muy específicos, integrándolos en la respuesta. Google Cloud ofrece **Web Grounding for Enterprise**, una variante de esta funcionalidad diseñada para entornos empresariales regulados, donde el modelo usa un subconjunto curado de contenido web con controles de cumplimiento y privacidad reforzados [20]. Esto es útil, por ejemplo, para industrias como finanzas, salud o sector público que requieren que las consultas web se limiten a fuentes confiables y que no salga información sensible.

¿Cómo se activa el grounding web? Al realizar la llamada al modelo (vía API o SDK), se debe incluir una indicación de que queremos habilitar la búsqueda web. En el **Generative AI API** de Vertex, esto se logra añadiendo la herramienta `enterpriseWebSearch` en la solicitud. Como vimos en el ejemplo anterior, en Python se hizo mediante `tools=[Tool(enterprise_web_search=EnterpriseWebSearch())]` al configurar la generación [17]. En la interfaz REST JSON equivaldría a incluir un campo `"tools": [ { "enterpriseWebSearch": {} } ]` en el cuerpo de la petición [21].

Al hacer esto, **Gemini 2.5 Flash realizará búsquedas en internet** sobre la marcha para obtener datos pertinentes a la pregunta o tarea, y luego generará su respuesta **basándose en esa información encontrada**. De esta manera, obtenemos respuestas **actualizadas y fundamentadas en fuentes externas**, reduciendo al mínimo las alucinaciones o desactualizaciones. Por ejemplo, si se pregunta *"¿Cuál es la población actual de tal ciudad?"*, el modelo buscará esa información en la web y la incorporará. Internamente, Vertex AI se encarga de manejar la búsqueda de forma segura: en el caso de Enterprise Web Grounding, se garantiza que la búsqueda y procesamiento de resultados ocurre dentro de regiones específicas (US/UE) y que no se loguea data sensible del usuario [20] [22].

Importante: El uso de grounding web puede implicar latencias ligeramente mayores (por el tiempo de búsqueda) y consumo de cuotas de búsqueda. Además, está **desactivado por defecto**; solo se activa si explícitamente se incluye la herramienta de búsqueda en la petición [23]. Asegúrese de usarlo únicamente cuando el caso de uso lo requiera (por ejemplo, para enriquecer la respuesta con datos que el modelo base por sí solo no conoce o no recuerda con precisión). En nuestro proyecto, podría ser útil habilitarlo si queremos que el agente proporcione datos adicionales no presentes en los documentos, o valide la información extraída consultando fuentes oficiales en la web.

Protocolo Agent-to-Agent (A2A) y agentes en Vertex AI

El protocolo **Agent-to-Agent (A2A)** es una iniciativa abierta impulsada por Google para estandarizar cómo distintos agentes de IA se comunican y colaboran entre sí [24]. En sistemas complejos, puede haber múltiples agentes o servicios de IA especializados (por ejemplo, uno encargado de extraer datos de documentos, otro experto en responder preguntas generales, etc.). A2A define un *lenguaje común* para que agentes de distintas plataformas o vendors puedan interoperar. **Usando A2A, los agentes publican sus capacidades y negocian cómo interactuar** – por ejemplo, qué formato de mensajes usan, si la interacción será vía texto, formularios o incluso audio/vídeo – todo de forma segura y estandarizada [25].

En el contexto de Google Cloud Vertex AI, A2A cobra vida a través de herramientas como el **Agent Development Kit (ADK)** y el **Agent Engine**:

- **Agent Development Kit (ADK):** Es un framework de código abierto (actualmente disponible en Python) que simplifica la construcción de agentes inteligentes y **sistemas multi-agente** [26]. Con ADK, puedes definir agentes con pocas líneas de código, especificando su lógica, sus herramientas (por ejemplo, llamar a Document AI como una función, o realizar una búsqueda web) y la forma en que razonan. ADK permite controlar de forma determinista el comportamiento de los agentes, imponiendo *guardrails* (límites y reglas) para garantizar que sigan ciertas políticas o pasos lógicos [27] [28]. También facilita interacciones más naturales con los agentes, incluso con capacidades de audio/video en tiempo real para crear asistentes más humanos si se requiere [29]. Un punto clave es que **ADK soporta múltiples modelos y herramientas**: puedes usar Gemini (está optimizado para ello) u otros modelos de la *Model Garden*, e integrar conectores a datos de tu empresa, APIs, etc., incluidos conectores MCP (Model Context Protocol) o herramientas de frameworks externos como LangChain [30]. Es decir, ADK te permite **orquestar un agente** que combine lo mejor de varios mundos (LLM, datos propios, llamadas a Document AI, búsquedas web, ejecución de código, etc.) con control preciso.

- **Vertex AI Agent Engine:** Es el servicio administrado donde puedes **desplegar agentes** ya construidos de forma escalable y con controles empresariales. Te quita la carga de manejar infraestructura, sesiones de conversación, escalado, seguridad, monitoreo, etc., para que pases de prototipo a producción sin reescribir todo [31] [32]. El Agent Engine se integra con ADK (u otros frameworks como LangChain, Crew.ai, etc.), permitiendo desplegar cualquier agente desarrollado con esas herramientas [33]. Ofrece características como *memoria de corto y largo plazo* (los agentes recuerdan contexto de interacciones previas) [34], evaluaciones automáticas de calidad, almacenamiento de ejemplos (Example Store) para mejorar iterativamente el agente, e integración con **Google Agentspace** (un hub empresarial para compartir agentes dentro de la organización) [35]. Además, impone medidas de seguridad: filtrado de contenidos, control de identidades y permisos con que opera el agente, *service controls* para evitar fugas de datos, validación de herramientas antes de ejecutarlas, y trazabilidad completa de cada paso que toma el agente [36] [37]. Todo esto es fundamental si nuestro agente multimodal va a usarse en un entorno real con datos sensibles.

¿Cómo encaja A2A en nuestra solución? En el proyecto descrito, se menciona un "agente A2A" que se conectará a GCP para usar Document AI y Gemini Flash. Esto sugiere que diseñaremos nuestro sistema como **un agente principal orquestador** que utiliza varias habilidades: por un lado la habilidad de *visión/documento* (extraer texto de imágenes con Document AI, verificar imágenes con Vision AI), y por otro la habilidad de *LLM* (analizar información, responder preguntas utilizando Gemini 2.5). Gracias al protocolo A2A y las herramientas de agentes de Vertex, podemos estructurar la solución de la siguiente manera (conceptualmente):

- El agente principal podría estar construido con ADK en Python, definiendo pasos o sub-agentes. Por ejemplo, podríamos definir un sub-agente o función llamada "ExtractInfoFromImage" que internamente llame a la API de Document AI para procesar una imagen/PDF; y otro sub-agente "AnswerQuestion" que use el modelo generativo Gemini para elaborar una respuesta usando tanto la consulta del usuario como la información extraída. Estas dos partes se comunican de forma estándar (el agente principal le pasa la imagen al sub-agente extractor, recibe de éste el texto extraído, y luego se lo proporciona al sub-agente de LLM junto con la pregunta del usuario). Con

A2A, **cada sub-agente puede publicar su capacidad** (por ejemplo: *"puedo extraer texto de documentos"* o *"puedo responder preguntas en lenguaje natural"*) y el protocolo maneja la interacción entre ellos de forma consistente [25].

- Alternativamente, podríamos implementar la orquestación manualmente en Python sin usar múltiples agentes formales: simplemente el flujo de código hace primero la llamada a Document AI y luego la llamada a Gemini. Esto también es válido; sin embargo, si queremos escalabilidad y modularidad (por ejemplo, en el futuro agregar más componentes, como un agente que busque en base de datos interna, etc.), adoptar el enfoque multi-agente con A2A/ADK desde ahora puede ser beneficioso. De hecho, Google reporta casos donde empresas construyen **sistemas de 2 o 3 agentes especializados que colaboran**: uno enfocado en obtener/consultar datos, otro en aplicar reglas de negocio o cálculos, y otro en interactuar con el usuario final [38]. Esa separación mejora la mantenibilidad y permite que cada agente esté optimizado en su tarea específica.

- Usando **ADK**, podríamos definir *herramientas* para el agente de LLM. Por ejemplo, definir una herramienta o función Python accesible al agente que, dada una imagen o PDF, utiliza Document AI para devolver texto. Luego, durante una conversación, el modelo Gemini podría invocar esa herramienta cuando necesite analizar una imagen (esto sería posible gracias a la capacidad de *function calling* que soporta Gemini Flash [39]). Es decir, el propio LLM podría decidir: "Tengo una imagen, voy a usar la herramienta `ExtractInfoFromImage`" y ADK se encargaría de ejecutar esa función y retornar el resultado al modelo. Todo esto se alinea con A2A/MCP, permitiendo integrar servicios externos (como Document AI via API) dentro del flujo conversacional del agente [40].

En resumen, A2A y las herramientas de Vertex AI Agents nos proveen el marco para **conectar múltiples componentes de IA de forma cohesiva**. En nuestra implementación, esto significa que el agente podrá: recibir una entrada (por ejemplo, un PDF con imágenes y texto), delegar la extracción de texto a Document AI, opcionalmente verificar la imagen con Vision AI, luego pasar la información relevante al modelo generativo Gemini 2.5 Flash (posiblemente utilizando grounding web para datos adicionales), y finalmente entregar al usuario una respuesta o resultado. Todo orquestado bajo un flujo controlado, seguro y reproducible.

Flujo de trabajo del agente e integración con el front-end

Finalmente, es útil visualizar el **flujo completo** de la solución propuesta, integrando los servicios mencionados:

1. **Recepción de la solicitud (prompt multimodal):** El agente (por ejemplo, una aplicación Python corriendo en Cloud Run o en el Agent Engine) recibe del usuario una consulta. Esta consulta podría incluir texto y archivos adjuntos (por ejemplo: "¿Cuál es el número de parcela y dueño registrado de este predio?" + una imagen/PDF catastral). En una arquitectura A2A, este sería el mensaje de usuario que ingresa al agente principal. En una arquitectura manual, simplemente es la entrada manejada por nuestro código backend (p. ej., vía una API REST).

2. **Verificación y pre-procesamiento (opcional):** El agente puede primero verificar que la entrada es adecuada. Aquí encaja el **verificador Vision AI** mencionado: por ejemplo, usar Cloud Vision para

asegurar que la imagen adjunta es legible y relevante (no está en blanco, no es una foto irrelevante, etc.), o para extraer metadatos rápidos. Esto es opcional pero agrega robustez.

3. **Extracción de texto con Document AI:** El agente pasa la imagen/PDF al **procesador de Document AI** correspondiente (usando la técnica descrita anteriormente). Obtiene de vuelta el texto extraído y posiblemente estructuras como tablas o campos detectados. Este contenido se guarda en variables o en la memoria del agente para usarlo en el siguiente paso. (En un entorno multi-agente, aquí el agente principal podría delegar a un sub-agente OCR, el cual retorna el documento procesado).

4. **Composición del prompt para el modelo generativo:** Ahora el agente tiene tanto la pregunta del usuario como la información relevante extraída de la imagen. Estos se combinan para formar el **prompt** que se enviará al modelo *Gemini 2.5 Flash*. Por ejemplo, el prompt podría ser: *"El usuario proporcionó la siguiente información extraída de un documento: [texto del documento]. Con base en esto, responde a la pregunta: [pregunta del usuario]."* También podemos incluir instrucciones específicas en el *prompt* (usando system o developer messages) para guiar al modelo, por ejemplo: *"Si no encuentras ciertos datos en el texto, responde que no está disponible"* o *"Proporciona la respuesta en formato JSON con campos X, Y, Z"*, etc., aprovechando las **instrucciones de sistema** que Gemini soporta [41] para formatear la salida según necesitemos.

5. **Llamada al modelo Gemini 2.5 Flash (con o sin grounding):** El agente envía el prompt compuesto al modelo generativo mediante Vertex AI. Aquí decidimos si activar el **grounding web**. Si la pregunta puede requerir información externa o actualización (por ejemplo, *"¿Qué valor catastral tiene actualmente esta propiedad?"* – quizás necesite datos externos), podemos activar la herramienta de búsqueda web en la llamada [17] . El modelo entonces genera la respuesta, posiblemente tras realizar búsquedas. El resultado es un **texto** que responde la consulta del usuario, posiblemente citando fuentes o proporcionando datos concretos. Toda la interacción con el modelo se hace vía la API (o ADK si estamos en ese marco), como vimos en los ejemplos de código.

6. **Post-proceso de la respuesta:** El texto devuelto por el modelo puede aún ser procesado antes de entregarlo. Por ejemplo, podríamos extraer del texto alguna referencia, o darle formato (si el front-end lo requiere en Markdown u otro), o aplicar un filtro de seguridad final. En Vertex AI Agent Engine, hay funcionalidades de post-procesamiento y validación que se pueden configurar como guardrails (por ejemplo, validar que el modelo no devuelva información prohibida). En nuestro caso, podríamos revisar que la respuesta contenga los datos clave (número de parcela, etc.) y no contenga información confidencial indebida.

7. **Respuesta al usuario (front-end):** El agente envía la respuesta final al front-end. Dado que se mencionó que posiblemente habrá un front en Next.js (Vercel) en el futuro, la comunicación seguramente será vía una API REST o WebSocket. Es decir, el backend (agente) ofrecerá un endpoint que el front consumirá. Gracias a la estructura en Markdown de esta respuesta, el front podría incluso renderizar formateado si se devuelve así, o simplemente mostrar texto plano/json según convenga. Integrar Next.js sería cuestión de consumir esta API del agente. *Por ahora, con nuestro agente funcionando en Python, podemos probarlo desde la línea de comandos o mediante peticiones HTTP con herramientas como cURL o Postman, antes de conectarlo al front.*

En conclusión, mediante la **integración de Document AI y Vertex AI (Gemini 2.5 Flash)** dentro de un **agente A2A**, obtenemos una solución poderosa: capaz de **entender documentos visuales** (imágenes

catastrales en PDF), **razonar** sobre la información obtenida con ayuda de un modelo de lenguaje avanzado y **enriquecer las respuestas** con información fresca de la web cuando sea necesario. Todo ello aprovechando la infraestructura de GCP, que proporciona escalabilidad, seguridad y herramientas especializadas para cada parte del problema. Esta documentación detallada sirve como guía para implementar paso a paso estos componentes en el proyecto actual, asegurando que el agente multimodal funcione de manera eficiente y correcta.

Fuentes: La información y pasos descritos se basan en la documentación oficial de Google Cloud y anuncios recientes. Para más detalles, puedes consultar la documentación de **Document AI** (procesamiento de documentos) [1] [3], la guía de **Gemini 2.5 Flash en Vertex AI** [12] [11], el tutorial de **Web Grounding (búsqueda web empresarial)** [42] [17], y el blog de Google Cloud sobre **Vertex AI Agents y el protocolo A2A** [24] [25], entre otros recursos mencionados a lo largo del texto. Estas referencias proporcionan mayor profundidad sobre cada servicio y cómo adaptarlos a las necesidades específicas de tu agente. ¡Con esta base, estarás listo para implementar la solución de análisis de imágenes catastrales con IA de manera exitosa! [1] [11]

---

[1] [2] [3] [4] Streamline Data Extraction with Google's Document AI | by Monish Mo | | Medium
https://medium.com/@monishmoz1/streamline-data-extraction-with-googles-document-ai-4c92fe280344

[5] [6] [7] [8] [9] [10] Quickstart: Process documents by using client libraries | Document AI | Google Cloud
https://cloud.google.com/document-ai/docs/process-documents-client-libraries

[11] [12] [13] [14] [18] [19] [39] [41] Gemini 2.5 Flash | Generative AI on Vertex AI | Google Cloud
https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-flash

[15] [16] [17] [20] [21] [22] [42] Web Grounding for Enterprise | Generative AI on Vertex AI | Google Cloud
https://cloud.google.com/vertex-ai/generative-ai/docs/grounding/web-grounding-enterprise

[23] Can gemini 2.0 flash thinking model use web search? : r/GeminiAI
https://www.reddit.com/r/GeminiAI/comments/1jpl1gu/can_gemini_20_flash_thinking_model_use_web_search/

[24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [40] Build and manage multi-system agents with Vertex AI | Google Cloud Blog
https://cloud.google.com/blog/products/ai-machine-learning/build-and-manage-multi-system-agents-with-vertex-ai