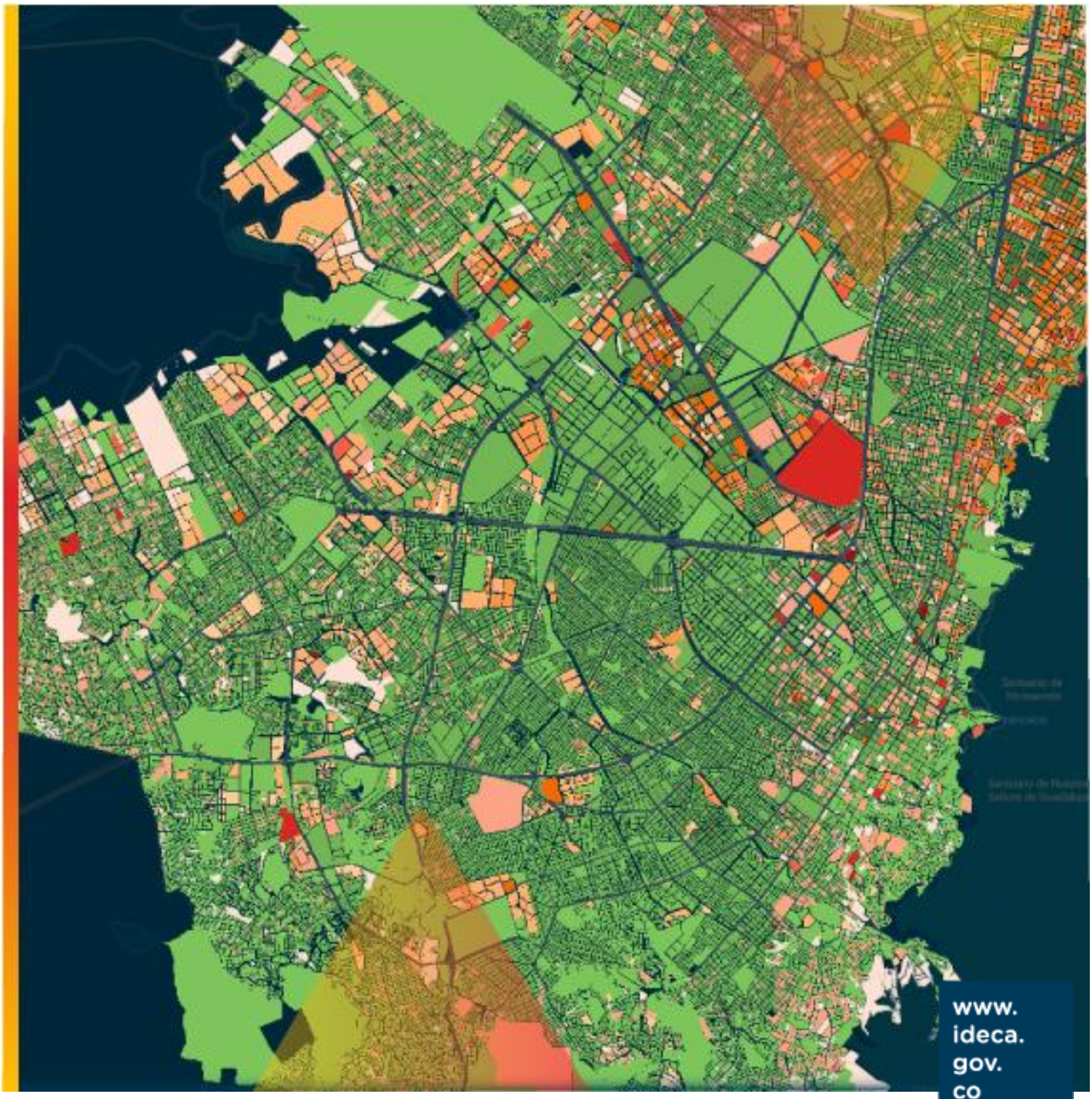
ideca



Clasificación del Uso Predial Usando Imágenes de GSV y Modelos de Lenguaje Visual

Bogotá, D.C., 04/08/2026

ALCALDÍA MAYOR DE BOGOTÁ D.C.

ideca

Clasificación del Uso Predial Usando Imágenes de GSV y Modelos de Lenguaje Visual

Gerente IDECA
Salomé Naranjo Luján

Subgerente de Analítica de Datos
Diego Ricardo Ibarra Rodríguez

Equipo de Trabajo

Contratista
Victor Alfonso Arias Vanegas

Fecha de creación o actualización: dd/mm/aaaa

Página web: www.ideca.gov.co

Correo electrónico: ideca@catastrobogota.gov.co

Licencia: Attribution 4.0 International (CC BY 4.0)

Control de Versiones

CAMBIOS

Fecha	Autor	Versión	Cambio efectuado
04/08/2026	Victor Alfonso Arias Vanegas	1.0	Primera versión del documento. No hay cambios para registrar.

REVISORES

Nombre	Dependencia
Diego Ricardo Ibarra Rodríguez	Subgerencia de Analítica de Datos

Tabla de Contenido

Resumen ejecutivo 5
Introducción 7
Antecedentes 7
Alcance 8
Objetivos 9
Inicio 10
Requerimientos 11
Estudios de mercado 11
Idear 12
Proceso de Ideación 12
Entendimiento de los Datos 12
Ingeniería de Características 14
Plataforma Tecnológica 15
Requisitos / Objetivos 16
Propuesta 16
Revisión con el cliente 17
Actividades 18
Prototipar 19
Prototipo / Modelado 19
Versión 1.0 20
Testeo 1.0 21
Incidencias 1.0 22

Ajustes 23

Revisión Cliente 23

Operacionalizar - Industrializar 24

Transferencia 25

Retrospectiva 27

Finalización 28

Actividades y análisis 28

Resultados 29

Conclusiones 29

Bibliografía (ICONTEC) 30

Lista de tablas

Tabla 1. Clases definidas en la taxonomía operacional 12

Tabla 2. Capas de la arquitectura y sus responsabilidades 15

Tabla 3. Fases del proyecto completadas y en planificación 18

Tabla 4. Evolución de los prompts de clasificación (p001–p004) 19

Tabla 5. Métricas de la evaluación online con el prompt p003 21

Table 6. Decisiones válidas de un revisor manual 24

Tabla 7. Detalle de los recursos de Azure desplegados 25

Lista de gráficas

Figura 1. Distribución de edificios por clase en el golden set de evaluación (55 muestras) 14

Figura 2. Arquitectura hexagonal: interacción entre las capas de la solución 17

Figura 3. Lógica de selección del adaptador VLM al arranque. 17

Figura 4. Diagrama de arquitectura e infraestructura de componentes en Microsoft Azure 25

Figura 5. Secuencia y flujo de dependencias de recursos del script de despliegue (deploy.sh) 26

Resumen ejecutivo

La aplicación clasifica el tipo de uso de una edificación en Bogotá a partir de varias imágenes de su fachada tomadas a nivel de calle. La unidad de análisis es el edificio: el modelo recibe entre una y cuatro vistas y produce una predicción a nivel de predio. Para cada caso entrega:

1. Clase primaria dentro de una taxonomía operacional de diez clases.
2. Hasta tres clases secundarias, especialmente para edificaciones de uso mixto.
3. Evidencia visual en español que sustenta la predicción.
4. Valor de confianza (0–1) por clase.
5. Indicador de revisión humana para casos de baja confianza, ambigüedad o clases sugeridas.
6. Trazabilidad completa en base de datos: modelo, prompt, schema, taxonomía y versión, y si aplica, la corrección humana.

La solución utiliza Qwen2.5-VL-32B-Instruct, un modelo de lenguaje visual servido mediante un Managed Online Endpoint de Azure ML / Microsoft Foundry sobre una GPU A100 de 80 GB de VRAM. Una aplicación FastAPI ofrece las interfaces de clasificación, revisión humana y procesamiento por lotes, y persiste cada predicción en una base de datos SQLite (con capacidad de migrar a PostgreSQL).

El proyecto ha demostrado un flujo de clasificación de extremo a extremo y una primera evaluación sobre 55 edificios. El baseline alcanzó 85,5 % de exactitud, Macro F1 de 0,661 y 100 % de respuestas JSON parseables. Las fases 00 a 06 del plan de 12 fases están completadas y los gates de decisión G0 a G3 fueron superados. La documentación respalda un MVP funcional y evaluado de forma inicial.

Introducción

La correcta identificación del uso que se le da a un predio dentro del tejido urbano de una ciudad es un insumo fundamental para la planificación territorial, la gestión catastral y la toma de decisiones públicas. En Bogotá, la actualización de esta información depende en gran medida de procesos manuales de verificación en campo, costosos y de baja frecuencia.

Este proyecto surge del interés por aprovechar los avances recientes en inteligencia artificial, particularmente los modelos de lenguaje visual multimodales (VLM), para automatizar la clasificación del uso predial a partir de imágenes de fachada tomadas a nivel de calle (Google Street View).

La primera versión del caso de uso (junio de 2025) demostró la viabilidad de la idea con el modelo Kimi-VL-A3B-Thinking y una taxonomía de 20 grupos de uso, desplegada como API REST. La versión actualizada que describe este documento migra la solución a una arquitectura cloud sobre Azure ML / Microsoft Foundry, consolida la taxonomía en 10 clases operacionales validadas con expertos catastrales y agrega evaluación formal, revisión humana y procesamiento por lotes.

Este documento es la memoria técnica de la versión actualizada del sistema y sigue la metodología documental de la entidad: describe el inicio, la ideación, la construcción del prototipo y la operacionalización de la solución, junto con los resultados, las conclusiones y la bibliografía de referencia.

Antecedentes

El interés por este tema se reforzó al explorar el estado del arte en visión por computador aplicado a entornos urbanos. La revisión SOTA de IDECA (2026) y el estudio de soluciones comparables en ciudades como Ámsterdam, Nueva York o Seúl muestran que la clasificación de usos a partir de imágenes de calle es una línea activa de innovación en la gestión territorial.

La línea base técnica (baseline) del proyecto se documentó en mayo de 2025 con la implementación Kimi + Outlines: un servicio FastAPI que clasificaba imágenes con el modelo moonshotai/Kimi-VL-A3B-Thinking-2506 y forzaba salidas estructuradas JSON mediante Outlines y Pydantic. Esa implementación operaba con la taxonomía legacy de 20 clases y producía listas de 3 a 5 categorías por imagen.

La evaluación del baseline arrojó 9.588 predicciones sobre 5.257 predios, con un marcado sesgo hacia la clase Residenciales (9.395 predicciones, ≈ 98 %) y una confianza promedio de 0,93 no calibrada. Estos hallazgos motivaron las decisiones D1 a D8 de la nueva fase: reducir la taxonomía a 10 clases, definir la unidad de análisis como el edificio y tratar la confianza como no calibrada.

Las lecciones reutilizables del baseline (estructura FastAPI, validación Pydantic, preservación del formato de imagen, patrón de carga "lazy" del modelo, exportación CSV) se integraron en el diseño de la versión actual, mientras que las piezas a reemplazar (esquema de salida, modelo de datos por edificio, cola de revisión) fueron rediseñadas desde cero.

Alcance

El sistema busca estimar el uso operacional de una edificación a partir de la evidencia visible en su fachada. La predicción pretende apoyar procesos de análisis y revisión, especialmente aquellos encargados de filtrar zonas de interés por cambios de uso entre un año y otro; no reemplaza una certificación catastral, una visita de campo ni una decisión administrativa oficial. Esta distinción es fundamental: el label produce una aproximación operacional del uso oficial a partir de lo que es visible en la fachada, no una declaración catastral definitiva.

El sistema recibe un identificador de edificio y entre una y cuatro imágenes. Analiza conjuntamente las vistas y produce una predicción a nivel de edificio. Válida que la respuesta del modelo cumpla el esquema JSON esperado, aplica reglas para determinar si el caso requiere revisión humana, persiste predicciones, evidencia, versiones, latencia y decisiones de revisión, y expone capacidades mediante API y tres espacios de trabajo web: Classify para clasificación interactiva, Review para la cola de revisión humana y Batch para trabajos masivos.

El sistema no certifica el uso catastral oficial de un inmueble. No garantiza el desempeño de producción para las diez clases con la evidencia actual. No opera con autenticación completa de usuarios ni autorización por roles. No afirma el cumplimiento legal sobre imágenes de Street View ni sobre políticas de retención.

Objetivos

Objetivo General

Desarrollar una solución automatizada basada en inteligencia artificial que permita clasificar el uso predial de los inmuebles de Bogotá a partir de imágenes de fachada, desplegada sobre una arquitectura cloud escalable en Azure, con evaluación formal, revisión humana y trazabilidad completa.

Objetivos Específicos

- Consolidar una taxonomía operacional de 10 clases de uso validada con expertos catastrales, junto con los contratos de datos versionados (Fase 01).
- Desplegar un endpoint administrado de VLM (Qwen2.5-VL-32B) sobre Azure ML / Microsoft Foundry (Fase 02).
- Diseñar prompts versionados y un pipeline de enforcement de esquema JSON con reparación y reintento (Fase 03).
- Construir el ciclo MVP de clasificación de un edificio, con persistencia y tres interfaces web (Fase 04).
- Implementar la revisión humana (HITL) con cola de auditoría y decisiones documentadas (Fase 05).
- Habilitar el procesamiento por lotes de hasta 1.000 edificios mediante Azure ML Batch Endpoint (Fase 06).

1. Inicio

La iniciativa surge como respuesta a una necesidad prioritaria identificada en el marco de la gestión catastral del Distrito Capital: la clasificación del uso del suelo predial requiere insumos actualizados, y los procesos manuales no logran cubrir la totalidad del territorio con la frecuencia requerida.

El caso de uso se formuló con base en el siguiente requerimiento funcional: ¿cómo automatizar la clasificación del uso del suelo predial en Bogotá, utilizando imágenes urbanas e inteligencia artificial, para mejorar la eficiencia y la actualidad de la información catastral?

2. Requerimientos

Durante la fase inicial se llevaron a cabo reuniones técnicas entre el equipo de desarrollo, expertos catastrales y áreas funcionales. De este proceso se derivaron los siguientes requisitos mínimos viables:

- Utilizar múltiples imágenes de fachada en tamaño real (entre 1 y 4) por cada predio para mejorar la precisión del modelo.
- Clasificar automáticamente los predios en los 10 grupos funcionales definidos por expertos, reduciendo la complejidad y el desbalance del universo de clases.
- Generar salidas interpretables y estructuradas bajo un formato estándar tipo JSON schema, con evidencia visual en español.
- Enrutar automáticamente a revisión humana los casos de baja confianza, ambigüedad, clases sugeridas o fallos de formato.
- Registrar trazabilidad completa de cada predicción (modelo, perfil, prompt, schema y taxonomía) para auditoría y reproducibilidad.
- Implementar el sistema como una API REST interoperable, adaptable a entornos locales, en contenedores o en la nube de Azure.
- Soportar procesamiento por lotes de hasta 1.000 edificios mediante Azure ML Batch Endpoint.

3. Idear

3.1. Proceso de Ideación

El proyecto se planificó con un esquema de 12 fases con gates de decisión, definido en DECISION_GATE_PLAN.md. Cada fase tiene un archivo detallado en phases/ con pasos, verificación y criterios de salida. Las fases 05 y 06 corren en paralelo tras la fase 04; la fase 07 requiere ambas.

La hipótesis central se mantuvo desde la versión alfa: «es posible clasificar automáticamente el uso predial de un inmueble urbano en Bogotá utilizando imágenes de fachada provenientes de Google Street View y modelos de lenguaje visual de razonamiento».

3.2. Entendimiento de los Datos

La unidad de etiquetado y predicción es el edificio. Generalmente, por experiencia, una única imagen es proporcionada, aunque el contrato admite entre una y cuatro imágenes. La salida representa una aproximación operacional del uso a partir de la fachada visible. Esta decisión fue formalizada como D2 en el ledger de decisiones: el label es una aproximación operacional del uso oficial catastral derivado de la evidencia visual, no una declaración catastral.

La taxonomía activa consta de diez clases operacionales, definidas en specs/taxonomy_operational.yaml y documentadas en dominio/taxonomia.docx. Seis clases están confirmadas y cuatro tienen restricciones de revisión obligatoria por ser sugeridas (es decir, una vez predichas, igualmente se le pide a un revisor manual una verificación), transitorias o por tener desempeño crítico en el baseline.

Tabla 1. Clases definidas en la taxonomía operacional

Clase	Interpretación resumida	Estado / tratamiento
RESIDENCIAL_1	Vivienda	Confirmada
COMERCIAL_1	Locales de compra y venta	Confirmada
COMERCIAL_2	Oficinas y servicios profesionales	Confirmada
COMERCIAL_3	Hoteles, moteles y restaurantes	Sugerida; revisión obligatoria
DOTACIONAL_1	Servicios comunales, salud, religión y recreación	Confirmada
DOTACIONAL_2	Educación	Sugerida; revisión obligatoria
MOLES_1	Construcciones grandes en obra	Sugerida/transitoria; revisión obligatoria
RURAL_1	Estructuras rurales o agropecuarias	Confirmada; sin muestras en baseline
MIXTO_1	Vivienda y comercio minorista	Confirmada
MIXTO_2	Vivienda y oficinas	Confirmada; desempeño baseline crítico

Las clases sugeridas (*COMERCIAL_3*, *DOTACIONAL_2* y *MOLES_1*) se predicen normalmente, pero siempre enrutan a revisión manual, según la decisión D3. *MOLES_1* es una clase de producción con revisión obligatoria, no un flag, según D1. Adicionalmente, existen tres flags de revisión que no son clases: *UNKNOWN_OR_INSUFFICIENT_EVIDENCE*, *REQUIRES_FIELD_VISIT* y *LOW_IMAGE_QUALITY*, estos se pueden producir como alternativa a una de las 10 clases predichas y permiten un mejor debugging en los ajustes del sistema.

El universo de predios presenta un fuerte desbalance: muy pocos tipos de uso agrupan la mayor cantidad de predios. Para la evaluación se construyó un golden set de 55 edificios que cubre 8 de las 10 clases, con muestras balanceadas por clase:

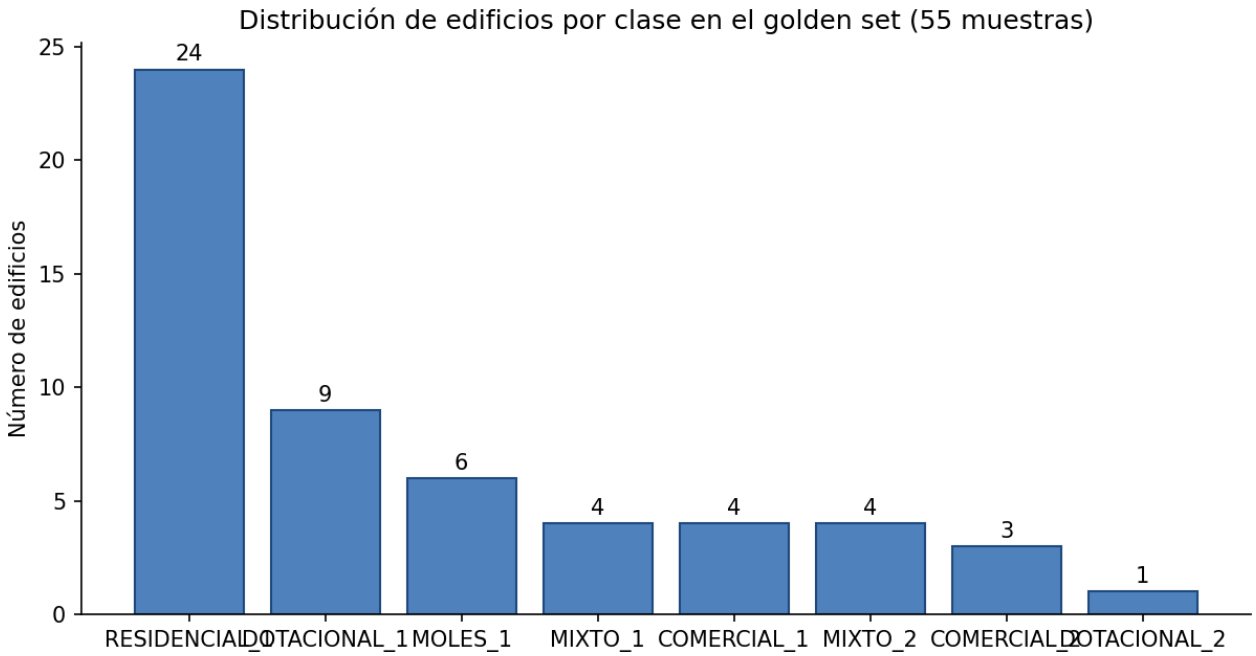


Figura 1. Distribución de edificios por clase en el golden set de evaluación (55 muestras)

Las clases *COMERCIAL_3* (hoteles, moteles y restaurantes) y *RURAL_1* no cuentan aún con imágenes de referencia de expertos y no son evaluables; *DOTACIONAL_2* cuenta con una sola muestra. Ampliar y balancear el golden set es una prioridad para trabajos futuros.

La taxonomía legacy de 20 clases se mapeó a la nueva taxonomía de 10 clases: las clases *Bodegas, Parqueaderos, Especiales* y *Mixto3* quedaron fuera y se resuelven por apariencia o con revisión de campo (*REQUIRES_FIELD_VISIT*); los antiguos Dotacionales 3, 4 y 5 fueron absorbidos por *DOTACIONAL_1*.

3.3. Ingeniería de Características

La etapa de ingeniería de características se enfocó en garantizar entradas homogéneas y trazables para el VLM:

Estandarización: las imágenes se convierten a RGB y se codifican como data URL JPEG; la dimensión máxima se controla con la variable IMAGE_MAX_DIM.

Multientrada: entre 1 y 4 imágenes por edificio, enviadas juntas en un solo llamado al VLM.

Flags de calidad: oclusión, desenfoque, nocturna, fachada ausente, estado de construcción, obstrucción severa y duplicado cercano; la baja calidad no bloquea la clasificación, solo se registra.

Un hallazgo de la evaluación evidenció que la reducción de la resolución a 400 px es insuficiente para detectar placas profesionales (indicador clave de MIXTO_2).

Contratos versionados: la documentación define esquemas legibles por máquina en specs/, que incluyen `taxonomy_operational.yaml`, `classification_output_schema.json`, `batch_input_schema.json`, `prediction_record_schema.json`, `review_record_schema.json` y `review_routing.yaml`.

3.4. Plataforma Tecnológica

La solución se ejecuta sobre el siguiente stack tecnológico:

Tabla 2. Capas de la arquitectura y sus responsabilidades

Capa	Módulo	Responsabilidad
Delivery	src/api/	HTTP, UI inline, rutas, middleware, lifecycle del endpoint
Aplicación	src/application/	Orquestación de flujos (classify, review), reglas de negocio
Puertos	src/ports/	Interfaces abstractas (VlmClientPort)
Adaptadores	src/adapters/	Implementaciones del puerto (Foundry, Mock)
Parsers	src/parsers/	Extracción + validación + repair de JSON del VLM
Prompts	src/prompts/	Builders de prompt versionados
Persistencia	src/persistence/	ORM, repositorios, engine/session, migraciones
Infraestructura	src/infrastructure/	Clientes del SDK de Azure ML (online/batch)
Utils	src/utils/	Utilidades compartidas (image encoding)

Persistencia con SQLAlchemy 2.0 asíncrono (SQLite por defecto, PostgreSQL-ready); infraestructura en Azure Container Apps, Azure ML / Foundry y Azure Container Registry; empaquetado con uv y Python ≥ 3.11.

3.5. Requisitos / Objetivos

Con base en los hallazgos y aclaraciones del cliente, los requisitos mínimos viables acordados se listan en la sección 1.2, y los objetivos específicos derivados en la sección de Objetivos. Ambos están directamente alineados con las fases del plan.

3.6. Propuesta

El sistema usa arquitectura hexagonal (puertos y adaptadores) para aislar la lógica de dominio de los detalles de infraestructura (Azure, VLM, base de datos). Esto permite: modo mock sin costo, intercambio de proveedor cloud y testeo del dominio sin GPU (local).

La solución adopta una arquitectura hexagonal. Los servicios de aplicación dependen de un puerto abstracto (`VlmClientPort`) para el cliente VLM, mientras los adaptadores concretos implementan la conexión con Foundry o un modo mock. Esta separación facilita pruebas, demos sin costo y cambios de proveedor con menor impacto sobre el dominio de negocio.

Las capas de la solución y sus responsabilidades son las siguientes. La capa de ***Delivery*** ofrece la API HTTP, las interfaces web, el middleware y el ciclo de vida del endpoint. La capa de ***Aplicación*** orquesta la clasificación y la revisión. Los ***Puertos*** definen interfaces abstractas, en particular `VlmClientPort`. Los ***Adaptadores*** implementan el puerto: `FoundryVlmAdapter` para inferencia real y `MockVlmAdapter` para respuestas sin costo. El ***Dominio*** contiene los prompts, los parsers, la taxonomía y las reglas. La ***Persistencia*** gestiona los repositorios y los modelos de predicciones y jobs. La ***Infraestructura*** contiene los clientes y la administración de Azure ML.

La regla de inversión de dependencias establece que los módulos de aplicación, parsers y prompts nunca importan los adaptadores, la infraestructura ni el SDK de Azure. Los adaptadores implementan los puertos; el dominio no sabe qué adaptador corre.

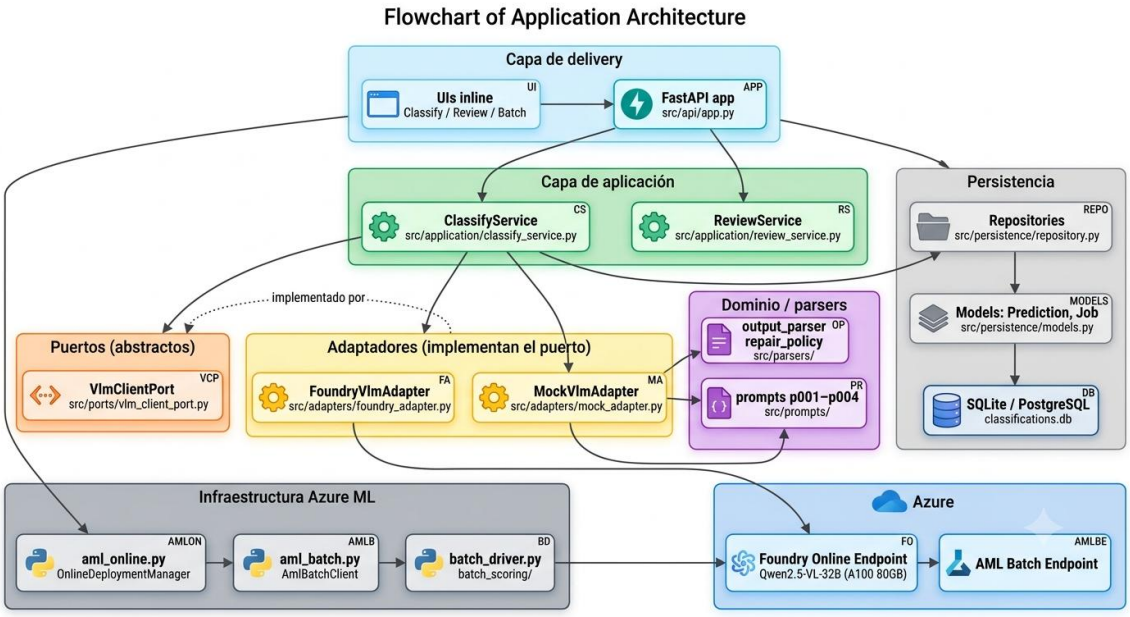


Figura 2. Arquitectura hexagonal: interacción entre las capas de la solución

La aplicación opera en modo real (invocando el endpoint Foundry) o en modo mock (FORCE_MOCK=1), con respuestas determinísticas sin costo de GPU. Si Foundry falla en el arranque, la aplicación cae a mock y se autorrecupera cuando el endpoint vuelve a estar disponible.

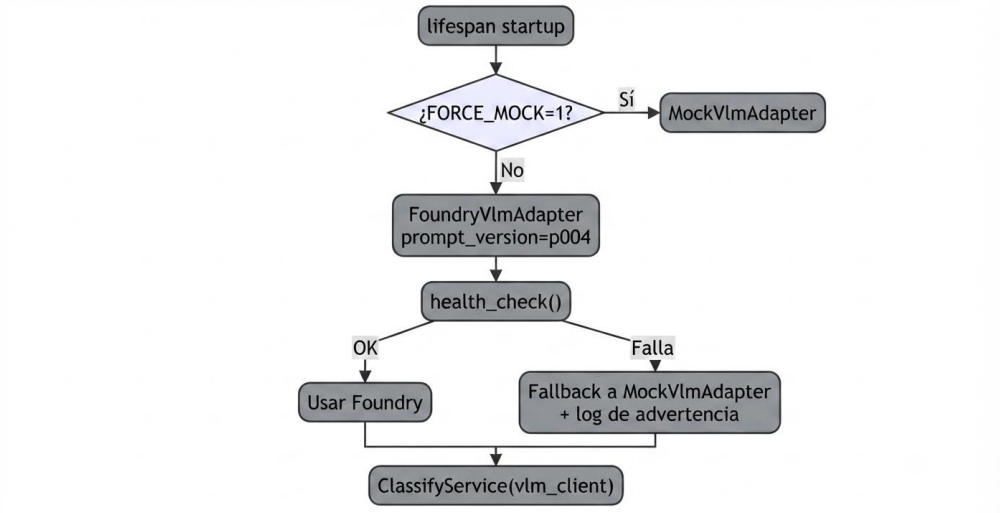


Figura 3. Lógica de selección del adaptador VLM al arranque (modo mock vs modo real).

3.7. Revisión con el cliente

La propuesta fue validada con expertos catastrales en iteraciones de revisión. Los hallazgos de las fases 03 y 05 (separación comercio vs. uso mixto, invisibilidad de placas profesionales) se incorporaron directamente al prompt p004, y se acordó que las correcciones de los revisores serán candidatas a datos de entrenamiento (decisión D29), sin ingresar automáticamente al pool.

También se acordó la revisión humana simple por defecto (D26), la restricción de etiquetas a la taxonomía de 10 clases (D28) y la ruta de revisión obligatoria para clases sugeridas (D3).

3.8. Actividades

Las principales actividades programadas y ejecutadas se organizan en 12 fases con gates de decisión:

TABLA 3. Fases del proyecto completadas y en planificación

Fase	Objetivo	Estado
00	Preparación del entorno Azure	Completada
01	Taxonomía y contratos	Completada G0
02	Endpoint Foundry	Completada G1
03	Prompt y enforcement de schema	Completada G2
04	MVP de clasificación	Completada G3
05	Harness y baseline de evaluación	Completada
06	Revisión humana (HITL)	Completada
07	Procesamiento batch	Completada

08	Benchmark de modelos	TODO
09	Calibración y umbrales	TODO
10	Endurecimiento de producción	TODO
11	Decisión de fine-tuning	TODO

4. Prototipar

4.1. Prototipo / Modelado

La selección del modelo se basó en la capacidad de razonamiento visual de los VLM de la familia Qwen2.5-VL y en la disponibilidad del catálogo Hugging Face de Microsoft Foundry. El perfil activo (80gb_high_accuracy) usa Qwen2.5-VL-32B-Instruct con temperature 0.0 y límite de 2.048 tokens de salida.

Los perfiles alternativos quedaron documentados: 24gb_fast (Qwen3-VL-8B, InternVL3-8B u otros cuantizados a 4 bits; SKU A10 denegada por Microsoft) y api_baseline (no implementado).

Los prompts evolucionaron de forma versionada desde p001 hasta p004 (decisión D24), sin editar versiones anteriores y almacenando la versión usada en cada predicción:

Tabla 4. Evolución de los prompts de clasificación (p001–p004)

Versión	Cambios clave	Resultado
p001_v0	Histórico; sin inspección vertical ni regla CRITICAL MIXTO_1	Conservado para reproducir evaluaciones
p001	Taxonomía 11 clases (10 + UNKNOWN); requisito «ALL floors»; regla CRITICAL MIXTO_1	Parse 100 % (Fase 03, 2026-06-10)
p002	Refinamiento COMERCIAL_1/MIXTO_1 por feedback de stakeholders: letreros y flujo de clientes = comercio; rejas metálicas NO	—
p003	Análisis por piso (analisis_por_piso), tabla de decisión y bands de confianza	Baseline: 85,5 % accuracy, macro F1 0,661 (Fase 05)

p004	Activo. Fix MIXTO_2: indicadores secundarios de oficina (AC splits, persianas uniformes, puertas de vidrio, directorio); distinción MIXTO_2 vs COMERCIAL_2	Pendiente de re-evaluación (Fase 08)

La aplicación aplica un pipeline de extracción, validación, reparación y reintento. El parser extrae el JSON de la respuesta del modelo, lo valida contra el esquema (usando pydantic), y si falla intenta reparaciones básicas (eliminar code fences, comas trailing, recortar al JSON más externo) antes de un reintento con un mensaje de corrección.

El número máximo de reintentos es uno. Entre las reglas de validación se incluyen: campos obligatorios, etiquetas válidas dentro de las diez clases, confianza entre 0 y 1, entre una y tres clases, evidencia visual de uno a ocho ítems, y exactamente una clase primaria. Si la reparación falla, el caso se marca para revisión en lugar de retornar un error HTTP 500. Esta capa reduce el riesgo de que una respuesta libre del modelo llegue directamente a consumidores posteriores (versión no estructurada para un consumer a posteriori).

Las reglas de enrutamiento a revisión (`specs/review_routing.yaml`) se evalúan en orden de prioridad: confianza < 0,65; margen top-2 < 0,10; clase UNKNOWN; clases sugeridas (***COMERCIAL_3***, ***DOTACIONAL_2***, ***MOLES_1***); y fallo de parse. El flag ***LOW_IMAGE_QUALITY*** no enruta, solo se registra.

4.2. Versión 1.0

El flujo principal combina inferencia automática, controles deterministas y revisión humana. El modelo no decide de manera aislada: su salida se valida, se somete a reglas y se registra con contexto suficiente para auditoría. El proceso consta de seis pasos.

Primero, el usuario proporciona el *building_id* y las imágenes de fachada. Segundo, la aplicación valida la entrada y codifica las imágenes para enviarlas al VLM (Visual Language Model). Tercero, el modelo analiza las vistas y devuelve una clasificación estructurada. Cuarto, el parser extrae y valida el JSON; si es necesario, intenta repararlo y realizar un reintento (para obtener una estructura correcta). Quinto, las reglas de revisión consideran la confianza, el margen entre clases, las clases sugeridas, la calidad y los fallos de parsing. Sexto, el resultado se persiste en la base de datos y, si corresponde, se incorpora a la cola de revisión humana.

Diseño tolerante a fallos: Un error de formato del modelo no debería convertirse automáticamente en un error HTTP 500. El sistema intenta reparar la respuesta y, si no puede validarla, conserva el caso marcado para revisión humana.

La aplicación actúa como una SPA inline que sirve tres workspaces en el puerto 7860: Classify (clasificación individual de 1 a 4 imágenes), Review (cola de revisión humana con decisiones auditadas) y Batch (carga de ZIP, lanzamiento de jobs de Azure ML, monitoreo, importación y exportación).

4.3.Testeo 1.0

La primera evaluación completa se ejecutó el 14 de junio de 2026 con Qwen2.5-VL-32B-Instruct, perfil 80gb_high_accuracy y prompt p003. El golden set contiene 55 edificios y cubre ocho de las diez clases. Los gates mostrados son operacionales, definidos en evals/eval_config.yaml.

Tabla 5. Métricas de la evaluación online con el prompt p003

Métrica	Resultado	Gate operacional	Estado
Accuracy	85,5 % (47/55)	>= 0,80	**Pasó**
Macro F1	0,661	>= 0,60	**Pasó**

Weighted F1	0,822	>= 0,80	**Pasó**
Parse rate	100 %	>= 0,90	**Pasó**
Latencia media	42,7 s	<= 120 s	**Pasó**

Que todas las compuertas operacionales hayan pasado significa que el prototipo cumplió los umbrales establecidos para esa evaluación. No significa que el sistema esté aprobado para producción. El golden set es un bootstrap y está muy lejos del rango mínimo planificado de 800 edificios, según la decisión D30.

Los hallazgos principales son los siguientes. ***MIXTO_2*** obtuvo 0 aciertos de 4: el modelo lo confundió con ***RESIDENCIAL_1***; este es el problema principal y motivó el prompt p004, que añade indicadores secundarios de oficinas. ***DOTACIONAL_2*** tuvo una sola muestra y fue confundido con ***DOTACIONAL_1***; el resultado no es concluyente. ***COMERCIAL_3*** y ***RURAL_1*** no tienen muestras en el golden set y no son evaluables. La confianza no está calibrada y no debe interpretarse como probabilidad fiable. El prompt p004 no ha sido reevaluado formalmente.

Operacionalmente, la evaluación tardó ~39 minutos (latencia media 42.7 s) y costó ~USD 2. 35 (0. 65 h × ~USD 3. 60/h). El parse de JSON fue 100 % exitoso sin reintentos y no se registraron errores de API.

4.4. Incidencias 1.0

MIXTO_2 sin aciertos (0/4): las placas profesionales, indicador clave de oficina en casa, son invisibles a la resolución de 400 px; el modelo confundió los casos con ***RESIDENCIAL_1*** (3) y ***MIXTO_1*** (1).

DOTACIONAL_2 con una sola muestra, clasificada como ***DOTACIONAL_1*** (F1 = 0); resultado no concluyente con n = 1.

Un caso RESIDENCIAL_1 fue clasificado como ***RURAL_1*** (confianza 0,85), clase sin muestras en el golden set.

Las clases ***COMERCIAL_3*** y ***RURAL_1*** no tienen imágenes de referencia y no son evaluables (gaps de evaluación).

Duplicados en los identificadores del golden set (p. ej. golden-MIXTO-001 con dos etiquetas distintas) apuntando a imágenes diferentes; pendiente de saneamiento.

Latencia media de 42.7 s por edificio: adecuada para revisión y batch, pero alta para una demo interactiva.

4.5. Ajustes

Prompt p004: indicadores secundarios de oficina (AC splits, persianas uniformes, puertas de vidrio, directorio, ausencia de elementos domésticos), tabla de decisión expandida y distinción ***MIXTO_2*** vs ***COMERCIAL_2***; pendiente de re-evaluación contra el golden set.

Pipeline de repair con reintento máximo 1 y persistencia del caso en la cola de revisión ante fallo de formato (nunca HTTP 500).

Reglas de enrutamiento priorizadas (confianza, margen, sugeridas, UNKNOWN, parse) implementadas en `ReviewService`.

Plan de experimentos para la Fase 08: ablación de resolución de imagen (IMAGE_MAX_DIM), número de vistas (1/2/4), comparación de modelos y prompts, y ampliación del golden set (objetivo 800+ edificios).

4.6. Revisión Cliente

La revisión humana se implementó como control de calidad integrado al flujo: los casos enrutados se atienden en la cola /review y las decisiones quedan documentadas con revisor, marca temporal y motivo.

Table 6. Decisiones válidas de un revisor manual

Decisión	**review_status resultante**	**Efecto**
accept	accepted	`requires_review=False`
change_label	corrected	Reescribe primary_label en JSON, `is_corrected=True`
mark_unknown	unknown	Label --> ***UNKNOWN_OR_INSUFFICIENT_EVIDENCE***
request_images	images_requested	Espera más imágenes
field_visit	field_visit	Requiere visita física

5. Operacionalizar - Industrializar

Durante la etapa de operacionalización se llevó a cabo una verificación integral del flujo: pruebas del endpoint /classify con imágenes reales, validación de la consistencia del JSON generado por el VLM, verificación de la interoperabilidad entre los módulos (adaptadores, parsers, servicios, repositorios) y del correcto preprocesamiento de imágenes.

Comportamientos verificados del flujo de clasificación: las llamadas al VLM son síncronas pero se envuelven en `asyncio.to_thread` para no bloquear el event loop; si el endpoint online no está corriendo, `POST /classify` devuelve 503; el reintento máximo es 1; los fallos de parse no devuelven 500 sino que marcan la predicción para revisión.

5.1. Transferencia

La aplicación se empaqueta en un contenedor Docker (python:3.11-slim, cómputo en CPU, sin GPU) y se despliega en Azure Container Apps mediante el script infra/deploy.sh, con Azure Container Registry para la imagen y un share de Azure Files para la persistencia de la base de datos.'

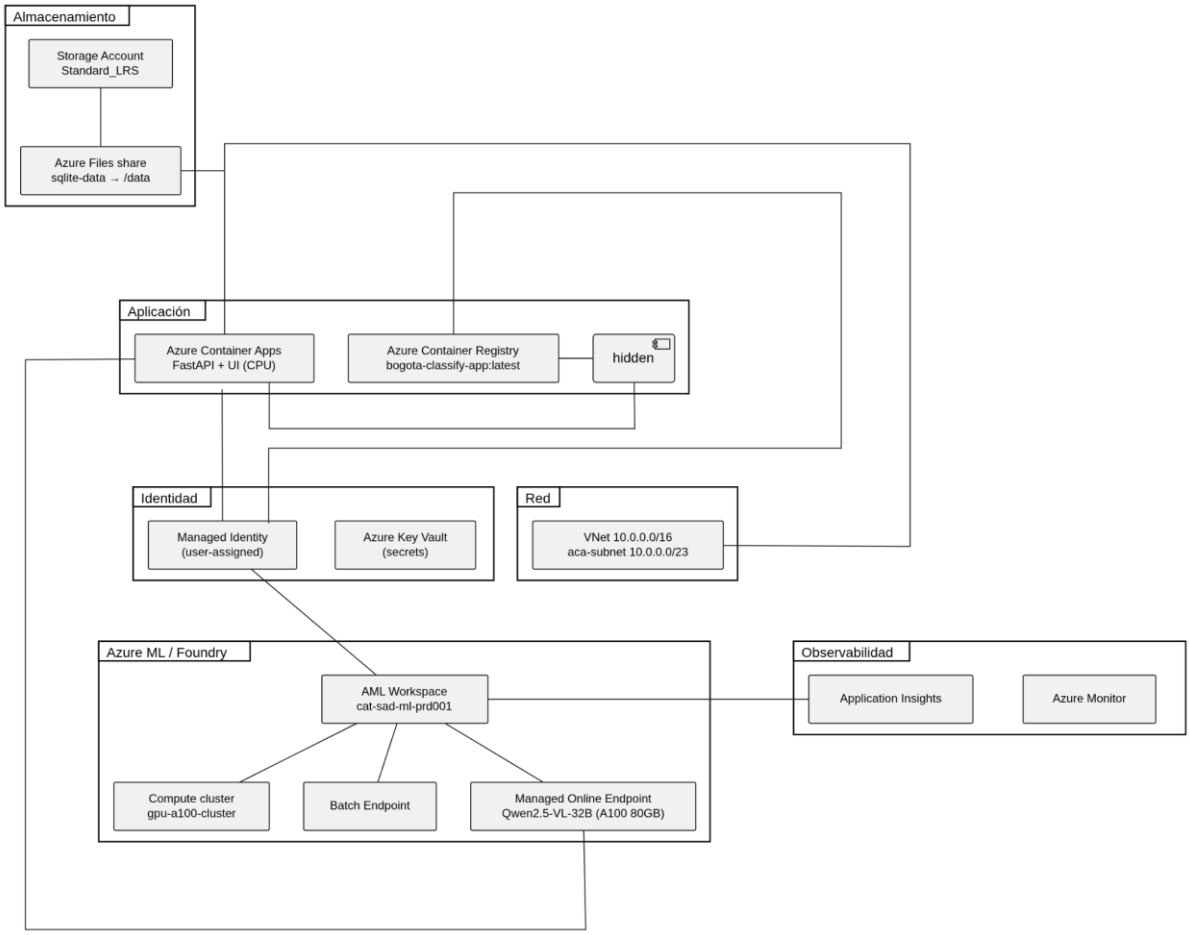


Figura 4. Diagrama de arquitectura e infraestructura de componentes en Microsoft Azure

Tabla 7. Detalle de los recursos de Azure desplegados

Recurso	**SKU / config**	**Propósito**
AML Workspace	*cat-sad-ml-prd001, eastus,* suscripción *az-sc-lz-online-catastro*	Hub de ML; Foundry, endpoints, compute

Managed Online Endpoint	Standard_NC24ads_A100_v4 (A100 80GB)	Serving del VLM (OpenAI-compatible)
Batch Endpoint	Compute cluster gpu-a100-cluster	Jobs de hasta ~1.000 edificios
Container Apps	1 replica, 1 CPU / 2 GiB, puerto 7860	Hosting de FastAPI + UI
Container Registry	Basic, admin disabled	Imagen bogota-classify-app:latest
Storage Account	Standard_LRS, StorageV2	Azure Files share para SQLite persistente
VNet	10.0.0.0/16, subnet 10.0.0.0/23	Aislamiento de red de Container Apps
Managed Identity	user-assigned	AcrPull + Azure ML Data Scientist
Key Vault	—	Secrets (referenciado; secrets runtime vía env vars)
Application Insights	vinculado al workspace	Logs/metrics

El script deploy.sh (~262 líneas) aprovisiona de forma imperativa: grupo de recursos, ACR, VNet con subnet dedicada, cuenta de almacenamiento, Managed Identity con roles AcrPull y Azure Machine Learning Data Scientist, el entorno de Container Apps y la Container App con restricción de IP. ***La infraestructura declarativa (Bicep/Terraform) está catalogada como deuda técnica.***

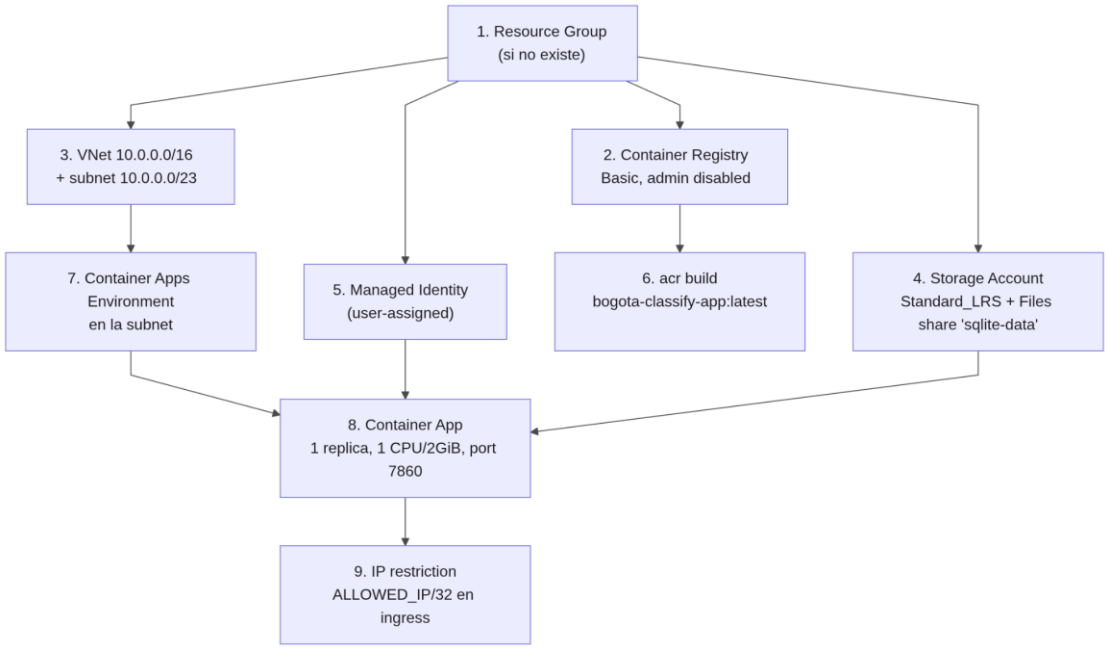


Figura 5. Secuencia y flujo de dependencias de recursos del script de despliegue (deploy.sh)

Los endpoints de Azure ML se gestionan de forma independiente: el online (Qwen2.5-VL-32B, API compatible con OpenAI, start/stop fire-and-forget, cold start de 10-15 mins, costo ~USD 3,67/h solo al correr) mediante `OnlineDeploymentManager` y *`scripts/deploy_foundry_endpoint.py`*; el batch (hasta ~1.000 edificios, ~USD 1,50 por job en 25 minutos) mediante `AmlBatchClient`, cuyo driver (b*`atch_scoring/code/batch_driver.py`*) rutea las predicciones al endpoint online con un prompt p004 embebido.

5.2. Retrospectiva

Lecciones aprendidas de la versión alfa aplicadas en la versión actual:

- La unidad de análisis debe ser el edificio, no la imagen: el modelo recibe varias vistas y produce una sola predicción por predio.

- El enforcement de esquema debe ser parte del pipeline (parse → validar → reparar → reintentar), no solo del prompt.
- La taxonomía debe validarse con expertos desde la Fase 01; la migración de 20 a 10 clases redujo la complejidad y el desbalance.
- La confianza del VLM no es calibrada: se trata como no calibrada hasta la Fase 09.
- Cargar el modelo localmente (trust_remote_code) se evitó al servirlo en el endpoint administrado de Foundry.
- El control de costos con start/stop del endpoint (costo solo por uso) es clave para un presupuesto de ~USD 35–45 mensuales.

Aspectos a mejorar:

- Golden set pequeño (55 edificios) y con duplicados; ampliar a 800+ con cobertura geográfica y de calidad de imagen.
- Monitoreo, runbook y revisión de seguridad pendientes (Fase 10).
- Infraestructura como código declarativa (Bicep/Terraform) pendiente.
- Sync manual del prompt p004 embebido en el batch driver.
- Latencia media alta (42,7 s) para una demo interactiva (Mejorar el desarrollo del frontend y del backend).
- Independencia en el deployment entre `/clasify` y `/batch` ya que ambos dependen de un mismo endpoint desplegado, y ese este se tiene que eliminar cada vez que un proceso interactivo termina, cuando se requiere hacer un job en batch no hay endpoint disponible y el resultado son errores.
- Aunque hay suficiente memoria VRAM, la imagen de Microsoft Foundry trae quemados 10k tokens como límite total de tokens, esto es un problema cuando se agregan mas de 4 imágenes como entrada al modelo VLM, se debe reconstruir esta imagen manualmente para evitar errores de sobrepaso de la ventana de contexto.

5.3. Finalización

Estado actual: las fases 00 a 06 están completadas y los gates G0 a G3 fueron superados: taxonomía y contratos cerrados (G0), endpoint Foundry funcional (G1), prompt p001 con 100 % de parse (G2) y loop de clasificación end-to-end (G3).

Actividades y análisis

El detalle de actividades por fase, con pasos, verificación y criterios de salida, se encuentra en phases/ (12 archivos) y en DECISION_GATE_PLAN.md. Las fases 00 a 06 fueron completadas con sus entregables verificados; las fases 07 a 11 permanecen en planificación y se ejecutarán secuencialmente, dependiendo de los gates de decisión G4, G5 y G6. La tabla resumen del estado del plan se presentó en la sección 2.8.

6. Resultados

Al cierre de las fases 00 a 06 se obtuvieron los siguientes resultados técnicos verificables:

- Flujo de clasificación end-to-end funcional con Qwen2.5-VL-32B-Instruct sobre Azure ML / Foundry (gates G1–G3 superados).
- Baseline de evaluación: 85,5 % de exactitud, macro F1 0,661, weighted F1 0,822, 100 % de respuestas parseables y 0 errores de API sobre 55 edificios.
- Tres workspaces operativos: Classify, Review y Batch, servidos por una única aplicación FastAPI.
- Procesamiento por lotes habilitado: jobs de hasta ~1.000 edificios por ~USD 1.50 en ~25 minutos.
- Latencia de clasificación: 7s (1 imagen) y 84s (4 imágenes); costo unitario ~USD 0.0084 por edificio.

7. Conclusiones

La tecnología de modelos de lenguaje visual aplicada a imágenes urbanas tiene alto potencial para modernizar los procesos catastrales del Distrito, entregando información que no solo predice, sino que explica.

La arquitectura hexagonal con puertos y adaptadores permitió operar con modo mock sin costo y aislar el dominio de la infraestructura cloud.

El baseline (85,5 % de exactitud, 0,661 de macro F1) supera los gates operacionales, pero está por debajo de los gates de lanzamiento propuestos (macro F1 ≥ 0,85); el problema principal es MIXTO_2, atacado con el prompt p004.

La revisión humana integrada y la trazabilidad completa hacen que cada predicción sea auditable y reproducible.

El enforcement de esquema con reparación y reintento garantiza respuestas estructuradas sin fallos de formato (100 % de parse en el baseline).

El control de costos con endpoints bajo demanda y facturación por uso hace viable la operación continua con un presupuesto modesto (~USD 35–45 mensuales).

El proyecto sigue una hoja de ruta clara de 12 fases con gates: completadas las fases 00–06, quedan benchmark, calibración, endurecimiento y la decisión de fine-tuning.

8. Bibliografía (ICONTEC)

CHEN, Shuyan; LI, Xiaodong; LIANG, Junhao. Kimi: An Open Multimodal LLM for Thinking. 1ª ed. arXiv: Cornell University, 2024. Disponible en: https://arxiv.org/abs/2507.03323 [Consulta: 4 de agosto de 2026].

QWEN TEAM. Qwen2.5-VL Technical Report. 1ª ed. arXiv: Cornell University, 2025. Disponible en: https://arxiv.org/abs/2502.13923 [Consulta: 4 de agosto de 2026].

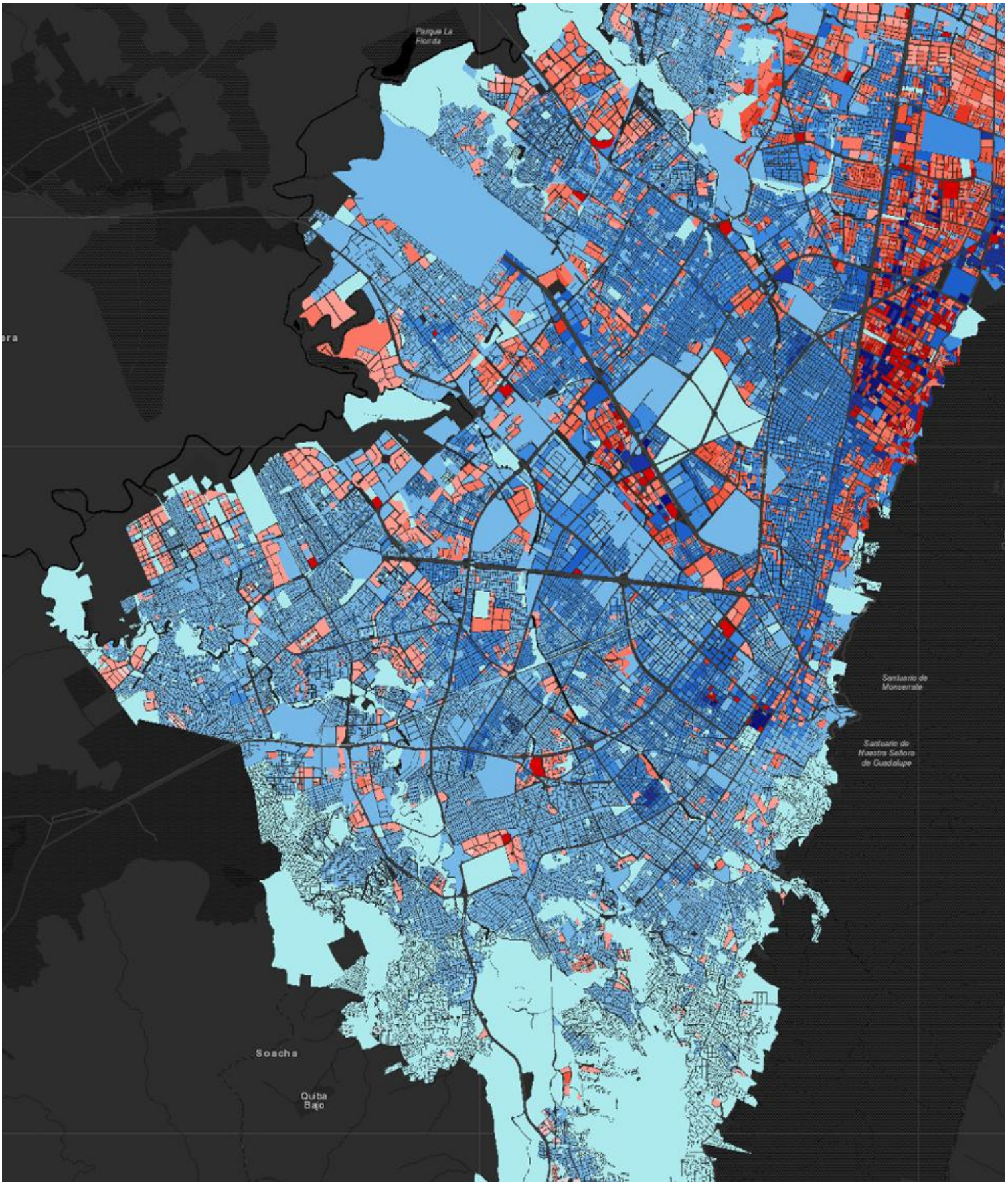
MOONSHOT AI. Kimi-VL-A3B-Thinking-2506 – Model Card. 1ª ed. París: Hugging Face, 2025. Disponible en: https://huggingface.co/moonshotai/Kimi-VL-A3B-Thinking-2506 [Consulta: 4 de agosto de 2026].

INSTITUTO GEOGRÁFICO AGUSTÍN CODAZZI – IGAC. Manual Técnico de Catastro Multipropósito. 2ª ed. Bogotá: IGAC, 2022. 214 p.

IDECA – Infraestructura de Datos Espaciales de Bogotá. Portal Oficial. 1ª ed. Bogotá: Alcaldía Mayor de Bogotá, 2025. Disponible en: https://www.ideca.gov.co [Consulta: 4 de agosto de 2026].

IDECA. Estado del arte: Modelos de Lenguaje Visual para la clasificación de tipos de uso en Bogotá. 1ª ed. Bogotá: IDECA, 2026.

MICROSOFT. Documentación de Azure Machine Learning y Microsoft Foundry. 1ª ed. Redmond: Microsoft, 2026. Disponible en: https://learn.microsoft.com/es-es/azure/machine-learning [Consulta: 4 de agosto de 2026].



ideca

ALCALDÍA MAYOR DE BOGOTÁ D.C.

ideca