Análisis de Factibilidad: Aplicación de Digitalización y Gestión con IA Local

1. Resumen de la Idea

Una aplicación de escritorio en **Python** que digitaliza notas físicas (papel), utiliza **IA local (Ollama)** para analizar, estructurar y expandir el contenido sin alterarlo, y presenta una interfaz gráfica (GUI) organizada por tiempos de implementación o cronología.

2. Puntuación de Factibilidad (Estado Actual)

A continuación, se presentan los scores (0-100) basados en la descripción inicial, evaluando la viabilidad técnica, usabilidad y complejidad.

Criterio	Score	Justificación
Factibilidad Técnica (General)	**85/100**	Python tiene librerías excelentes para todo lo requerido (GUI, Backend, API).
Procesamiento de Texto (IA)	**90/100**	Ollama facilita enormemente correr modelos como Llama 3 o Mistral localmente.
Digitalización (OCR Manuscrito)	**40/100**	**Punto Crítico.** Convertir letra manuscrita (handwriting) a texto digital es *mucho más difícil* que texto impreso. Tesseract suele fallar con letra humana normal.
Rendimiento (Hardware)	**60/100**	Correr IA localmente consume mucha RAM y GPU. Puede hacer que la app sea lenta o inusable en PCs promedio.
Interfaz de Usuario (UX)	**80/100**	Es totalmente realizable con librerías modernas de Python.
Privacidad de Datos	**100/100**	Al ser local, la privacidad es total, lo cual es un gran punto de venta.

Promedio General de Factibilidad: 75.8/100

3. Análisis de Puntos Débiles (Por qué el score bajó)

1. **El Cuello de Botella del OCR (Score 40):**
    - Tu idea depende de "trasladar notas de papel". Si las notas son impresas, es fácil. Si son manuscritas (lo más común en notas rápidas), las librerías estándar como `pytesseract` tienen un rendimiento muy pobre.

2. **Latencia de la IA Local (Score 60):**
    - Analizar texto, estructurarlo y darle puntajes de factibilidad con un modelo local requiere cómputo. Si el usuario no tiene tarjeta gráfica dedicada (NVIDIA), la app podría tardar minutos en procesar una sola nota, frustrando la experiencia.
3. **Rigidez del Prompt:**
    - Lograr que la IA *no* modifique la idea pero sí la profundice requiere una ingeniería de prompts muy precisa ("System Prompt Engineering"). Existe riesgo de alucinaciones (inventar datos).

4. Plan de Mejora e Integración de LLaVA

Para aumentar la factibilidad y la calidad del producto, implementaremos mejoras técnicas, enfocándonos específicamente en la integración de un modelo de Visión (VLM).

A. Análisis de Factibilidad: LLaVA (Large Language-and-Vision Assistant)

La inclusión de LLaVA a través de Ollama cambia las reglas del juego para la digitalización.

- **Impacto en Calidad (Positivo):** A diferencia de un OCR tradicional (que lee letra por letra), LLaVA "entiende" la imagen. Puede interpretar flechas, diagramas simples, listas desordenadas y mala caligrafía basándose en el contexto semántico.
- **Impacto en Rendimiento (Negativo):** LLaVA es pesado. Mientras que un OCR tarda milisegundos, LLaVA puede tardar de **5 a 30 segundos** por imagen dependiendo de la GPU del usuario.

Estrategia Híbrida Recomendada:

1. **Paso 1:** Intentar OCR rápido (EasyOCR). Si la confianza es alta (>85%), usar ese texto.
2. **Paso 2 (Fallback Inteligente):** Si el OCR falla o detecta "ruido", invocar LLaVA para "leer e interpretar" la imagen.
    - *Prompt para LLaVA:* "Transcribe textualmente el contenido manuscrito de esta nota. Mantén el formato de listas si existen."

B. Solución al Problema del OCR (De 40 -> 85 con LLaVA)

- **Mejora:** Implementar **LLaVA v1.6 (o versiones cuantizadas como `llava-phi3`)** en Ollama.
- **Plan B (Híbrido):** Incluir una función de "Dictado de Voz" (Whisper local). A veces es más rápido leer tu nota en voz alta que intentar que el OCR descifre una mala caligrafía.
- **Plan C (UX):** Crear un editor de corrección rápida post-escaneo. El usuario ve la foto de su nota y el texto extraído al lado para corregir errores del OCR antes de que la IA lo procese.

C. Optimización de IA Local (De 60 -> 85)

- **Modelos Cuantizados:** Usar obligatoriamente modelos cuantizados (4-bit) a través de Ollama (ej. `llama3:8b` o `phi3:mini`). `Phi-3` es excelente para análisis lógico y corre en laptops sin GPU potente.

- **Procesamiento Asíncrono:** La interfaz no debe congelarse mientras LLaVA procesa la imagen. Usar `threading` o `asyncio` en Python para mostrar una barra de progreso o permitir al usuario digitalizar la siguiente nota mientras la anterior se procesa.

D. Estructura del Prompt (Ingeniería de Prompt)

Para cumplir el requisito de "no modificar, solo profundizar", el prompt del sistema debe ser estricto.

Prompt Estructurado Sugerido:

> "Actúa como un Analista de Proyectos Senior. Tu tarea es tomar la siguiente nota cruda y analizarla. REGLAS ESTRICTAS:
>
> 1. NO cambies la idea central.
> 2. NO agregues características que no estén implícitas.
> 3. Tu salida debe ser un JSON con: 'Titulo_Generado', 'Resumen', 'Analisis_Factibilidad', 'Consideraciones_Tecnicas', 'Tiempo_Estimado_Implementacion' (Corto/Medio/Largo Plazo).
> 4. En el cuerpo, expande los puntos técnicos necesarios para que la idea funcione."

5. Stack Tecnológico Recomendado (Python)

Para lograr esto eficientemente, este es el "Blue Print" técnico actualizado:

1. **Interfaz Gráfica (GUI):**
   - Recomiendo `CustomTkinter` o `Flet`.
   - *Por qué:* `Tkinter` básico se ve antiguo. `CustomTkinter` ofrece modo oscuro y diseño moderno nativo.
2. **Backend IA (Multimodal):**
   - `Ollama`: Corriendo modelos de texto (`Llama3`, `Mistral`) y modelos de visión (`LLaVA`, `Moondream`).
   - **Nota sobre `Moondream`:** Es una alternativa a LLaVA mucho más ligera y rápida, diseñada específicamente para correr en dispositivos con recursos limitados (como Raspberry Pi o laptops viejas). Podría ser mejor opción que LLaVA si el usuario no tiene GPU dedicada.
3. **OCR (Digitalización - Capa Rápida):**
   - `EasyOCR`: Para escaneos rápidos de texto impreso o manuscrito muy claro.
   - `OpenCV`: Para pre-procesar la imagen (binarización, corrección de perspectiva).
4. **Base de Datos Local:**
   - `SQLite`: Para guardar las notas, los análisis generados y los estados. No requiere servidor.

6. Roadmap de Desarrollo (MVP)

1. **Fase 1: El Motor de Texto (Backend)**
    - Crear script Python que conecte con Ollama.
    - Probar prompts para asegurar que la IA estructura bien la info sin inventar.
2. **Fase 2: El Ojo (Visión)**
    - Integrar `Ollama` con el modelo `llava`.
    - Crear función de prueba: Enviar imagen -> Recibir descripción textual.
    - Comparar `LLaVA` vs `Moondream` para decidir cuál usar por defecto.
3. **Fase 3: La Interfaz (Frontend)**
    - Diseñar la vista de "Dashboard" (Tablero Kanban o Lista cronológica).
    - Crear la vista de "Detalle" (donde se despliega la profundización).
4. **Fase 4: Integración**
    - Unir GUI con lógica. Implementar hilos para que la app no se trabe durante la inferencia de visión.

7. Plan de Implementación para Cursor / AI Assistant

Esta sección está diseñada para ser copiada y pegada en tu entorno de desarrollo (Cursor, Windsurf) para guiar a la IA en la construcción del proyecto paso a paso.

Prompt Inicial (Contexto Global)

Copia esto al inicio de tu sesión en Cursor para establecer las reglas del proyecto.

> "Actúa como un Ingeniero de Software Senior experto en Python. Vamos a construir 'PaperToPlan', una app de escritorio para digitalizar notas manuscritas usando IA local.
>
> Stack Tecnológico:
>
> 1. **GUI:** CustomTkinter (Moderno, Dark Mode).
> 2. **IA Engine:** Ollama (local API). Usaremos 'llama3' para texto y 'llava' (o 'moondream') para visión.
> 3. **OCR:** EasyOCR + OpenCV para pre-procesamiento.
> 4. **DB:** SQLite nativo.
>
> Reglas de Arquitectura:
>
> - Código modular: Separa `gui/`, `backend/`, `database/`.
> - **Asincronía obligatoria:** Las llamadas a la IA (Ollama) y al OCR NUNCA deben congelar la interfaz gráfica. Usa `threading` o `asyncio`.
> - Manejo de errores robusto: Si Ollama no está corriendo, la app debe avisar, no crashear.

¿Entendido? Espera mi primera instrucción para el Módulo de Backend."

Prompt Fase 1: El Motor de Visión e IA (Backend)

Usa esto para generar la lógica antes de la interfaz.

"Crea el módulo `backend/ai_manager.py`. Necesito una clase `AIEngine` con dos métodos principales:

1. `analyze_text(text_content)`: Recibe texto crudo, lo envía a Ollama (Llama3) con un System Prompt que pida analizar factibilidad y tiempos. La respuesta debe ser un JSON parseable.
2. `extract_text_from_image(image_path)`: Implementa una estrategia híbrida.
   - Primero, intenta usar EasyOCR. Si la confianza promedio es > 80%, retorna ese texto.
   - Si es < 80% (letra difícil), envía la imagen a Ollama usando el modelo 'llava' con el prompt 'Transcribe este texto manuscrito literalmente'.

Incluye manejo de excepciones si el servicio de Ollama no responde."

Prompt Fase 2: Base de Datos

Para persistencia de datos.

"Crea el módulo `database/db_manager.py`. Necesitamos una tabla `notes` en SQLite con los campos:

- `id` (auto increment)
- `raw_text` (texto extraído)
- `ai_analysis` (JSON almacenado como texto)
- `status` (pending, processed)
- `implementation_time` (short, medium, long - extraído del análisis)
- `created_at`

Crea funciones para insertar nota, leer todas, y actualizar análisis."

Prompt Fase 3: Interfaz Gráfica (GUI Skeleton)

Para armar la visual.

"Ahora vamos a la GUI usando `CustomTkinter`. Crea `main.py` y la carpeta `gui/`. Estructura principal:

1. Barra lateral izquierda: Botón 'Nueva Nota' (abre diálogo de archivo), Filtros (Corto/Medio/Largo plazo).

2. Área principal: Una lista scrollable de tarjetas. Cada tarjeta representa una nota procesada, mostrando el Título (generado por IA) y una etiqueta de tiempo.
3. Panel derecho (oculto por defecto): Al hacer click en una tarjeta, se despliega el detalle con el análisis completo de la IA.

Genera el código base de la ventana y la navegación, sin la lógica de IA todavía."

Prompt Fase 4: Conexión Asíncrona (Wiring)

El paso más crítico para la UX.

"Integra el Backend con la GUI.

1. Cuando el usuario seleccione una imagen en 'Nueva Nota', lanza un `threading.Thread`.
2. Muestra una barra de progreso indeterminada en la GUI.
3. El hilo debe ejecutar `extract_text_from_image` y luego `analyze_text`.
4. Al finalizar, guarda en DB y actualiza la lista de la UI usando `after` (para no romper el