SELECCIÓN DEL MODELO DE LENGUAJE Y ANÁLISIS DEL CÓDIGO

Mikel Martinez, Ibai García, Haimar Álvarez e Iker Iglesias
09/12/2025

RESUMEN EJECUTIVO

El proyecto **IA Video Analyst Pro** consiste en el desarrollo e implementación de un **Agente de IA Multimodal** avanzado diseñado para la automatización del análisis y edición de contenidos deportivos. Utilizando Python y Streamlit, el sistema integra el modelo **Google Gemini 3 Pro Preview**, seleccionado estratégicamente tras un análisis comparativo por su superioridad en el procesamiento nativo de video y su extensa ventana de contexto frente a alternativas como Nvidia Nemotron. La herramienta se distingue por su capacidad de **"Uso de Herramientas" (Tool Use)**, permitiendo no solo la interacción conversacional en lenguaje natural, sino la ejecución autónoma de tareas técnicas complejas: desde la ingesta segura de videos de YouTube y la estructuración de eventos temporales, hasta el recorte automático de clips (*highlights*) mediante FFmpeg y el análisis táctico frame a frame, ofreciendo así una solución integral, persistente y escalable para la gestión de activos audiovisuales.

ÍNDICE

SELECCIÓN DEL MODELO DE LENGUAJE Y ANÁLISIS DEL CÓDIGO.....0
RESUMEN EJECUTIVO.....1
1. Selección del Modelo de Lenguaje y Análisis del Código..... 2
1.1. Modelo Seleccionado: Google Gemini 3 Pro Preview.....2
Justificación Técnica de la Elección.....3
Implementación en el Código.....3
1.2. Comparativa con Modelos Descartados..... 4
1. nvidia/nemotron-nano-12b-v2-vl:free (Versión Gratuita)..... 4
2. nvidia/nemotron-nano-12b-v2-vl (Versión Estándar).....4
1.3. Estructura y Flujo de la Conversación.....5
1.4. Diseño y Creación del System Prompt.....6
1.5. Configuración y Ajuste de Parámetros.....6
1.6. Pruebas y Depuración.....7
1.7. Implementación y Despliegue..... 8

1. Selección del Modelo de Lenguaje y Análisis del Código

1.1. Modelo Seleccionado: Google Gemini 3 Pro Preview

Para constituir el núcleo cognitivo del sistema **IA Video Analyst Pro**, se ha seleccionado el modelo **google/gemini-3-pro-preview**, integrándolo a través de la interfaz de programación de aplicaciones

(API) de OpenRouter. Esta decisión no fue arbitraria, sino el resultado de un análisis exhaustivo de las capacidades requeridas para el procesamiento de video complejo.

Justificación Técnica de la Elección

La decisión final se sustenta en tres pilares técnicos fundamentales donde este modelo demostró una superioridad notable frente a la competencia:

1. **Capacidad Multimodal Nativa (Video-to-Text):**
   A diferencia de los modelos multimodales tradicionales (LMMs) que simulan la visión de video extrayendo capturas de pantalla estáticas (frames) a intervalos regulares, Gemini 3 Pro posee una arquitectura diseñada nativamente para ingerir archivos de video y audio como un flujo continuo de datos. Esta característica es crítica para el análisis deportivo, ya que permite al modelo comprender la **causalidad** (entender qué acción previa desencadenó un gol) y la **dinámica temporal** (percibir la velocidad y trayectoria del balón), elementos que se pierden al analizar imágenes estáticas aisladas.
2. **Ventana de Contexto Masiva (1M+ Tokens):**
   El análisis de un partido de fútbol o un resumen extenso (10-15 minutos) genera una cantidad masiva de "tokens" visuales. La mayoría de los modelos del mercado poseen ventanas de contexto limitadas (8k, 32k o 128k tokens), lo que obligaría a recortar el video o perder información crítica. Gemini, con su ventana superior al millón de tokens, permite pasar el archivo de video completo en una sola solicitud. Esto garantiza que el modelo tenga "memoria" de todo el encuentro, pudiendo relacionar un evento del minuto 90 con una acción ocurrida en el minuto 5 sin pérdida de información por truncamiento.
3. **Razonamiento y OCR (Reconocimiento Óptico de Caracteres):**
   El modelo demostró una capacidad excepcional no solo para identificar objetos, sino para realizar OCR sobre el marcador del partido presente en la transmisión. Esta capacidad fue vital para correlacionar dos líneas de tiempo distintas: el **tiempo de juego** (lo que dice el reloj del árbitro en pantalla) y el **tiempo de video** (el segundo exacto del archivo `.mp4` donde ocurre la acción). Esta distinción es indispensable para enviar las coordenadas temporales exactas a la herramienta FFmpeg y realizar los recortes con precisión quirúrgica.

Implementación en el Código

La integración técnica se refleja en el script `ai_agent.py`. La invocación del modelo no se realiza mediante prompts de texto simples, sino a través de la construcción de un `payload` JSON complejo. Este objeto incluye el video codificado en base64 y las instrucciones de sistema, configurando parámetros específicos como el `response_format` para garantizar que la salida sea estructurada y procesable por la aplicación.

```
payload = {
    "model": config.MODELO_VISION,
    "messages": [{"role": "user", "content": [{"type": "text", "text": prompt_text}, {"type": "video_url", "video_url": {"url": data_url}}]}],
    "temperature": 0.1,
    "response_format": {"type": "json_object"}
}
```

1.2. Comparativa con Modelos Descartados

Durante la fase de investigación y desarrollo (I+D), se sometieron a pruebas de estrés varias alternativas del mercado. A continuación, se detallan los motivos técnicos por los cuales fueron descartadas.

1. nvidia/nemotron-nano-12b-v2-vl:free (Versión Gratuita)

Este modelo se evaluó inicialmente por su eficiencia y coste cero, pero presentó fallos estructurales para este caso de uso:

- **Fallo de Infraestructura (Ceguera a Internet):** Se detectó que este endpoint carece de la capacidad de realizar peticiones externas para descargar contenido. Al intentar suministrar URLs directas de YouTube, la API devolvía consistentemente un error 404 (*No endpoints found that support video URLs*), lo que obligaba a realizar la descarga y procesamiento del video en el lado del cliente, aumentando la latencia y la complejidad.
- **Colapso por Contexto (Payload too Large):** Al intentar solventar el punto anterior enviando el video codificado en Base64, el modelo fallaba sistemáticamente. La versión gratuita impone límites estrictos de cómputo y memoria (VRAM) en los servidores de inferencia, lo que impedía procesar la densidad de información de un video de 5 minutos, incluso reduciendo su calidad al mínimo.
- **Alucinaciones Temporales Severas:** En las pruebas aisladas donde se logró procesar fragmentos muy cortos (mediante compresión extrema a 1 FPS y baja resolución), el modelo demostró una incapacidad para mantener la coherencia temporal. Generaba *timestamps* imposibles (ej. "Minuto 341" en un video de 10 minutos) o inventaba descripciones de jugadas (alucinaciones), lo cual es inaceptable para una herramienta de análisis factual.

2. nvidia/nemotron-nano-12b-v2-vl (Versión Estándar)

Se evaluó la versión de pago esperando superar las limitaciones de infraestructura, pero persistieron limitaciones arquitectónicas:

- **Limitación Arquitectónica (12B Parámetros):** Con solo 12 billones de parámetros, este modelo cae en la categoría de "Small Language Models" (SLM). Aunque eficiente para tareas

de visión estática simple, carece de la profundidad de razonamiento necesaria para comprender la secuencia causa-efecto en un deporte dinámico.

- **Incapacidad de OCR Preciso:** Para la automatización de recortes con FFmpeg, la precisión en la lectura del marcador es innegociable. Nemotron fallaba frecuentemente al distinguir los números pequeños o borrosos del marcador, y confundía sistemáticamente el tiempo de juego con el tiempo de reproducción del archivo, resultando en clips de video erróneos o desfasados.
- **Compresión Destructiva Necesaria:** Debido a su menor capacidad de entrada, era necesario reducir drásticamente la calidad visual del video antes del envío. En un contexto deportivo (plano general), esto provocaba que elementos pequeños como el balón o los números de los dorsales se volvieran invisibles para la IA, impidiendo cualquier tipo de análisis táctico real.

1.3. Estructura y Flujo de la Conversación

El sistema no funciona como un chatbot pasivo, sino bajo una arquitectura de **Agente Autónomo con Uso de Herramientas (Tool Use)** simulado. El flujo lógico es el siguiente:

1. **Input del Usuario:** El usuario interactúa en lenguaje natural a través de la interfaz de chat (ej: *"Quiero ver el gol de Messi"*).
2. **Evaluación de Intención (Cerebro):** El LLM, guiado por el *System Prompt*, analiza semánticamente la petición para clasificarla en una de dos categorías: solicitud de **Información** (texto) o solicitud de **Visualización** (video).
3. **Ramificación Lógica:**
    - **Rama A (Texto):** Si la intención es informativa, el LLM consulta el JSON interno con el análisis del partido y genera una respuesta explicativa en lenguaje natural.
    - **Rama B (Acción):** Si la intención es visual, el LLM *no* responde al usuario directamente. En su lugar, genera una instrucción técnica en formato JSON estricto: `{ "accion": "cortar", "tiempo": "...", "duracion": "..." }`.
4. **Ejecución de Código (Manos):** El script de Python (`app.py`) actúa como el ejecutor. Intercepta la respuesta de la IA, detecta si contiene el JSON de acción y, en caso afirmativo, invoca a la librería **FFmpeg** en el servidor para procesar el archivo de video real.
5. **Respuesta Final:** El sistema presenta al usuario el resultado de la operación (el clip de video generado) acompañado de una confirmación textual, cerrando el ciclo de interacción.

```python
if matches:
    txt = ""
    for m in matches:
        try:
            cmd = json.loads(m)
            if cmd.get('accion') == 'cortar':
                placeholder.write(f"✂️ {cmd['descripcion']}...")

                # Auto-recovery
                p_vid = video_manager.obtener_ruta_video(vid_id)
                if not os.path.exists(p_vid):
                    url = st.session_state['datos_partido'].get("url_origen")
                    if url: video_manager.descargar_video(url, vid_id)

                r = video_manager.cortar_video_ffmpeg(vid_id, cmd['tiempo_video'], cmd['descripcion'])
                if r: videos.append((r, f"Clip: **{cmd['descripcion']}**"))
        except: txt = content
```

1.4. Diseño y Creación del System Prompt

La Ingeniería de Prompts (*Prompt Engineering*) fue crítica para orquestar el comportamiento del agente. El *System Prompt* (instrucciones maestras) evolucionó iterativamente:

- **Versión 1 (Básica):** Se utilizó una instrucción genérica: *"Analiza el video y responde preguntas"*.
    - *Resultado:* Fallo. El modelo era vago, divagaba y no proporcionaba los tiempos precisos necesarios para la edición de video.
- **Versión 2 (Estructurada con JSON):** Se instruyó explícitamente al modelo para distinguir entre `tiempo_partido` y `tiempo_video`.
    - *Resultado:* Mejora significativa en la precisión de datos, pero el modelo presentaba inestabilidad en el formato de salida, a veces mezclando código con texto conversacional.
- **Versión 3 (Final - Restrictiva):** Se implementaron reglas negativas ("constraints") y ejemplos ("few-shot prompting"). Se eliminaron las alucinaciones y se logró que el modelo actuara estrictamente como un motor de extracción de datos cuando era necesario, evitando inventar videos ante preguntas simples como el resultado del partido.

```python
system_prompt = f"""
Eres un analista deportivo experto. Tienes los datos del partido en este JSON:
{contexto_str}

SIGUE ESTAS REGLAS AL PIE DE LA LETRA:

1. SI EL USUARIO PIDE INFORMACIÓN (Texto):
   - Preguntas como: "¿Cómo quedó?", "¿Quién marcó?", "¿Hubo tarjetas?", "Resumen", "¿Cuál fue el resultado?".
   - RESPONDE SOLO CON TEXTO. Explica el resultado o el dato usando el JSON.
   - PROHIBIDO generar JSON de corte en este caso.

2. SI EL USUARIO PIDE VISUALIZAR (Video):
   - Solo si usa verbos explícitos como: "Quiero ver", "Enséñame", "Muestra", "Sácame un clip".
   - ENTONCES responde SOLO con el JSON: {{ "accion": "cortar", "tiempo_video": "MM:SS", "duracion": 15, "descripcion": "titulo_breve" }}
"""
```

1.5. Configuración y Ajuste de Parámetros

Para optimizar el rendimiento del modelo en un entorno de producción, se ajustaron los hiperparámetros de la API:

- **Temperatura (0.1):** Se configuró una temperatura cercana a cero. En tareas creativas se usan valores altos, pero para la extracción de datos (tiempos exactos) y la generación de comandos de máquina (JSON), se requiere **determinismo absoluto**. Esto minimiza el riesgo de que el modelo "invente" minutos o cambie el formato del JSON.
- **Response Format (** Se activó el modo JSON nativo de la API durante la fase de análisis inicial. Esto garantiza sintácticamente que la salida sea un objeto JSON válido, evitando errores de *parsing* en el script de Python que podrían romper el flujo de ejecución.
- **Resolución de Video:** Se optó estratégicamente por descargar la "peor calidad" disponible (`worst[ext=mp4]`) que mantenga la legibilidad. Esto reduce el peso del archivo Base64, disminuyendo drásticamente el consumo de tokens (coste), la latencia de red y el riesgo de *timeouts* en la API, sin afectar significativamente la capacidad del modelo para reconocer eventos generales.

1.6. Pruebas y Depuración

Se superaron tres desafíos técnicos principales durante las pruebas:

1. **Bloqueo de YouTube (Error 403/401):**
    - *Error:* yt-dlp era bloqueado por YouTube al detectar un bot.
    - *Solución:* Implementación de autenticación mediante cookies.txt y simulación de cliente Android en las cabeceras de la petición.

```
ydl_opts = {
    'format': 'worst[ext=mp4]/mp4',
    'outtmpl': ruta_destino,
    'quiet': True, 'overwrites': True, 'nocheckcertificate': True, 'ignoreerrors': True, 'no_warnings': True,
    'cookiefile': config.COOKIES_FILE,
    'extractor_args': {'youtube': {'player_client': ['android', 'web']}},
}
```

2. **Lectura de Respuestas Híbridas:**
    - *Problema:* Ocasionalmente, la IA respondía mezclando lenguaje natural con código (ej: *"Claro, aquí tienes el JSON: { ... }"*). Esto provocaba fallos al intentar parsear la respuesta completa como JSON.
    - *Solución Técnica:* Se implementó un algoritmo de limpieza basado en **Expresiones Regulares (Regex)** (`re.findall(r'\{.*?\}', ...)`). Este código escanea la respuesta de la IA y extrae quirúrgicamente solo los bloques que cumplen con la estructura JSON, descartando el texto "ruido" circundante.

```
    content = ai_agent.obtener_respuesta_chat(msgs)

    # Procesar
    matches = re.findall(r'\{.*?\}', content, re.DOTALL)
    videos = []
    txt = content
```

3. **Persistencia de Video:**
    - *Problema:* Al cambiar de chat entre diferentes partidos, el archivo de video temporal se sobrescribía, rompiendo la funcionalidad de recorte en los chats anteriores.
    - *Solución Técnica:* Se diseñó un sistema de gestión de archivos basado en IDs únicos (hash del video). Además, se programó una lógica de "auto-recuperación": si el usuario solicita un clip de un partido antiguo cuyo video fue borrado para ahorrar espacio, el sistema detecta la ausencia del archivo y lo vuelve a descargar automáticamente en segundo plano.

```
    # Auto-recovery
    ruta_mp4 = video_manager.obtener_ruta_video(vid_id)
    if not os.path.exists(ruta_mp4):
        url = datos_archivo.get("url_origen")
        if url:
            with st.spinner("📥 Recuperando video..."):
                video_manager.descargar_video(url, vid_id)
    st.rerun()
```

1.7. Implementación y Despliegue

El código final se consolidó en una aplicación web interactiva utilizando el framework **Streamlit** (`app.py`), facilitando un despliegue rápido y una interfaz amigable.

- **Entorno:** Python 3.12 gestionado mediante entornos virtuales (`venv`) para el aislamiento de dependencias.
- **Dependencias Clave:**
    - `streamlit`: Gestión del Frontend y la interfaz de usuario.
    - `yt-dlp`: Motor de descarga e ingesta de video.
    - `ffmpeg`: Motor de procesamiento y edición de video (backend).
    - `opencv-python`: Utilizado para la captura de frames estáticos en el análisis táctico.
- **Seguridad:** Se implementó la librería `python-dotenv` para la gestión de variables de entorno, asegurando que claves sensibles como la API Key no queden expuestas en el código fuente (Hardcoding).