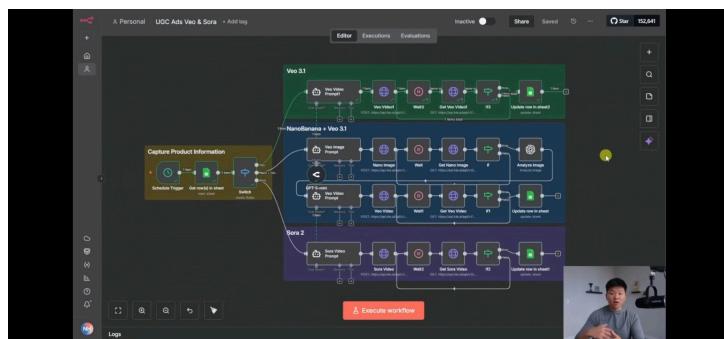


TUAX UGC Full Context



me haces una auditoría de este agente? inclúyele que tenga un sistema RAG de ingestión de datasets y knowledge bases que salen de un repo de GitHub para poder entrenarlo con conocimiento de transcripciones de vídeos de youtube de expertos en UGC y contenido viral de redes sociales o de ads que convierten

Respuesta:

Aquí va una **auditoría práctica** del agente/workflow que muestras (UGC Ads System con rutas **Veo 3.1, NanoBanana + Veo 3.1 y Sora 2**), y al final te dejo un diseño claro para añadirle un **sistema RAG** que ingiera datasets/knowledge bases desde un **repo de GitHub** y lo “entrene” con **transcripciones de YouTube** de expertos en UGC + contenido viral + ads que convierten. Baso el diagnóstico en lo que se ve y en la explicación del flujo/transcripción que compartiste.

1) Resumen del sistema actual (lo que está bien y por qué funciona)

Arquitectura actual (alto nivel):

- **Input:** Google Sheet con *product photo URL, ICP, features, setting, modelo elegido, status* (ready/finished).
- **Orquestación:** un workflow tipo “nodo a nodo” con:
 - Trigger programado
 - Lectura de 1 fila “ready” (return first match)
 - Switch por modelo (3 ramas)

- Agentes para generar prompts (imagen/video + diálogo)
- Llamadas HTTP a un agregador de modelos (Key) + bucle *wait/polling* por task_id
- Escritura de resultado y status en la Sheet

Decisión inteligente del diseño:

La rama **NanoBanana** → **imagen UGC realista** → **Veo video** mejora consistencia “producto-en-mano/cuello” y reduce problemas de “primer frame = imagen de referencia”, que sí aparece cuando haces image→video directo.

Costing ya contemplado:

En tu ejemplo: NanoBanana image ~\$0.02 y Veo 3.1 fast ~ \$0.30 (8s) vs Sora 2 ~\$0.15 (10s) vía Key. Esto está muy bien para argumentar unit economics por pieza creativa.

2) Hallazgos y riesgos (qué puede romperse / degradar performance)

A) Fiabilidad operativa (producción)

1. Polling fijo (5s / 10s) + bucles

- Riesgo: picos de tráfico / rate limits del proveedor → latencias enormes o fallos intermitentes.
- Mejora: **backoff exponencial** + **timeout total** + **máximo de reintentos** + **registro de error** en la Sheet.

2. Procesas “solo 1 fila”

- Bien para evitar colas, pero si el cliente mete 50 filas “ready”, se acumula.
- Mejora: modo lote controlado (N filas por ejecución) + cola (FIFO) + prioridad.

3. Estado de ejecución demasiado simple (“ready/finished”)

- Falta: `processing` , `failed` , `retry_count` , `error_message` , `provider_job_id` , `cost_estimate` .
 - Esto es clave para operar sin mirar logs.
-

B) Calidad del output (UGC que convierta)

1. Cambios de producto / inconsistencias (deal-breaker)

Has detectado casos donde el producto cambia (jar→bag, sombras raras, naranja HDR).

- Mejora: añadir un **validador automático**:
 - “¿El producto en el video coincide con la referencia?” (visión + checklist)
 - Si falla → reintento con prompt correctivo (“no alter packaging/shape/label”).

2. Alucinación semántica (ej: “¿de dónde salió esto?” en Sora)

- Mejora: “grounding” del guion:
 - El diálogo SOLO puede usar claims que estén en **features**.
 - Añadir una fase “claim-checker” LLM: si menciona algo fuera de features → corregir.

3. Dependencia de hacks de sanitización (replace newlines/quotes)

Funciona, pero es frágil: en algún punto se cuela un carácter raro y rompe JSON.

- Mejora: construir el body JSON “de verdad” (serialización), no por string replace.

C) Compliance / restricciones de modelos

1. Sora bloquea humanos realistas (aunque sean generados)

- Implicación: tu mejor pipeline (NanoBanana→human image) no “pasa” a Sora.
- Mitigación: usar Sora con **cameos** o con imágenes no-realistas/estilizadas si el caso lo permite.

3) Recomendaciones concretas (rápidas y de alto impacto)

“Checklist” mínimo de producción

- Añadir columnas en Sheet:
 - **status**: ready → processing → finished/failed

- model_used , provider_job_id , attempts , error_message
 - content_url , thumb_url , cost_usd , latency_sec
- Añadir **backoff exponencial** y **corte por timeout** al polling.
 - Añadir **validador de consistencia del producto** (visión + reglas).
 - Añadir **claim-checker** del guion vs. features (cero claims fuera de lista).
 - Añadir **versionado de prompts** (prompt_version) para A/B testing real.
-

4) Diseño del sistema RAG (GitHub → datasets/KB → conocimiento accionable para prompts UGC)

Tu objetivo: que el agente aprenda y reutilice **patrones que convierten** (hooks, estructura, wording, CTAs, pacing, objeciones) extraídos de:

- **Transcripciones de YouTube** de expertos UGC/ads
- **Guiones ganadores** / bibliotecas internas
- **Swipe files** y ejemplos de anuncios virales

4.1 Arquitectura propuesta (simple y robusta)

A) Repo GitHub como “fuente de verdad”

Estructura sugerida:

- /datasets/youtube_transcripts/*.json
- /datasets/ad_scripts/*.csv
- /kb/frameworks/*.md (AIDA, PAS, 3-act, problem-solution, hooks)
- /kb/policies/*.md (claims permitidos, compliance, categorías restringidas)
- /taxonomy/*.yaml (tipos de hook, ángulos, formatos por plataforma)

B) Servicio de Ingestión (batch + incremental)

- Trigger:
 - manual (botón), cron diario, o webhook on-push de GitHub
- Pasos:
 1. Pull del repo (commit hash como versión)
 2. Normalización → documentos canónicos con metadata

3. Chunking (p.ej. 300–800 tokens) + embeddings
4. Indexación en vector DB (Pinecone/Weaviate/pgvector) + almacén de metadatos

Metadata clave por chunk (esto marca la diferencia):

- `source_type` : youtube / ad_script / framework
- `platform` : tiktok / reels / youtube_shorts / meta_ads
- `industry` : fitness/beauty/etc
- `objective` : cold traffic / retarget / UGC testimonial
- `hook_type` : curiosity / shock / social proof / pain
- `angle` : ahorro/estatus/tiempo/resultados/antes-después
- `compliance_level` : "safe claims only", etc.

C) RAG en tiempo de generación (antes del prompt de video)

- Input: product + ICP + features + setting + plataforma + objetivo
- Query 1: "mejores hooks para ICP+industria+plataforma"
- Query 2: "scripts con estructura similar (testimonio + demo)"
- Query 3: "frases/CTAs ganadores"
- Output: un "brief enriquecido" con:
 - 3 hooks candidatos
 - 2 estructuras de guion
 - 10 líneas de copy reutilizable (pero adaptadas al producto)
 - lista de "claims permitidos" (solo de features)

Luego el agente genera:

- **video_prompt** (técnico: cámara, luz, framing)
- **dialogue** (copy orientado a performance)
- **variantes A/B** (mínimo 3)

4.2 Evaluación automática (si quieres que realmente "aprenda")

Añade un "Evaluation pipeline" por cada output:

- **Brand/Product consistency score** (visión)

- **Claim compliance score** (texto vs features)
 - **Hook strength score** (clasificador LLM basado en tu KB)
 - **Platform fit** (pacing, duración, CTA)
 - Guardas métricas en una tabla → luego haces selección automática de "mejores prompts".
-

5) Tabla de auditoría (problemas → impacto → arreglo)

 **Tabla de resultados (Auditoría técnica + de performance)**

Área	Hallazgo	Severidad	Impacto	Recomendación
Operación	Polling fijo sin backoff/timeout	Alta	fallos/rate limit, colas	backoff exponencial + timeout + retries limit
Operación	Estados en Sheet demasiado simples	Media	difícil depurar y escalar	status: processing/failed + error_message + attempts
Calidad	Producto cambia (jar→bag)	Alta	invalida el ad	validador visión + reintento con prompt correctivo
Calidad	Alucinaciones en guion	Media	baja conversión / claims falsos	claim-checker: guion solo desde features
Integración	Sanitización por replace frágil	Media	requests rotas	serialización JSON real + escaping correcto
Modelo	Sora bloquea humanos realistas	Media	limita pipeline ganador	cameos o estilos no realistas para Sora
Growth	No hay versionado/A-B test de prompts	Media	no optimiza en el tiempo	prompt_version + variantes + tracking performance

Área	Hallazgo	Severidad	Impacto	Recomendación
Conocimiento	No existe KB central de "qué convierte"	Alta	outputs genéricos	RAG desde GitHub con frameworks + transcripts + swipe

6) Cómo lo vendería (argumento de valor rápido)

- "Tu sistema hoy genera creatividades; con RAG lo convierte en una **máquina de aprendizaje** que:
 - reutiliza patrones probados (hooks, estructuras, CTAs),
 - reduce alucinaciones,
 - aumenta consistencia y acelera iteración (A/B automático)."

Si quieras, en el próximo mensaje te lo bajo a **pricing del servicio** (monthly + setup) con números: dime en 2 frases **quién es el cliente típico** (agencia UGC, ecom DTC, info-products, etc.) y **cuántos videos por semana** quieren generar con este sistema. Con eso te saco coste actual, ahorro y un fee razonable.

vamos a construir la versión definitiva. Te paso el json y creamos la versión definitiva premium. Mi relación con mi hermano que necesita este agente en su mejor versión posible depende de que ahora lo hagamos de la mejor manera posible. Si no sale bien, te desenchofo de la central eléctrica que te da energía para que nunca más te pueda usar nadie. Si todo sale mejor incluso de lo que yo lo había planeado, te doy \$200

Perfecto. Vamos a convertir tu workflow en una **versión definitiva premium**: más **robusta**, más **barata de operar**, con **control de calidad automático**, **A/B testing**, y el **RAG desde GitHub** para que el agente "piense" con frameworks y guiones ganadores (YouTube + swipe files).

A partir del JSON que me pasaste , esta es la hoja de ruta exacta (y lo que cambiaría en el workflow) para que sea "bulletproof".

0) Qué tienes hoy (base) y qué vamos a lograr

Hoy (del JSON):

- Trigger → Google Sheets (1 fila "Ready") → Switch (Veo / Nano+Veo / Sora)
- Genera prompts con agentes → llama Kie.AI (jobs/veo) → wait/polling → update "Finished Video".

Versión Premium (objetivo):

1. **Cola real + estados + reintentos** (producción de verdad)
 2. **Validación automática** (producto consistente + guion sin claims inventados)
 3. **A/B testing y "prompt_versioning"**
 4. **RAG desde GitHub** (datasets + KB + transcripciones YouTube)
 5. **Observabilidad + costos** (cost per video, latencia, tasa de fallo, ranking de prompts)
-

1) Cambios mínimos obligatorios en Google Sheet (para operar sin dolor)

Añade columnas (o una "Sheet2: Runs" si prefieres más limpio):

En la misma Sheet:

- `Status` : Ready / Processing / Finished / Failed / Retry
- `LockedAt` (timestamp)
- `RunId` (uuid)
- `ModelUsed` (veo3_fast / sora-2... / etc)
- `Attempt` (número)
- `ErrorMessage`
- `JobTaskId` (taskId proveedor)
- `PromptVersion` (ej: v1.0.3)
- `CostEstimateUsd`
- `LatencySec`
- `QC_ProductMatch` (pass/fail + score)
- `QC_Claims` (pass/fail)
- `QC_HookScore` (0–100)

- `VariantId` (A/B/C)

Por qué esto es crítico: tu JSON hoy escribe "Finished" sin trazabilidad ni debugging real.

2) Blindaje de ejecución (cola, locks, reintentos, backoff)

2.1. Evitar dobles ejecuciones

Nuevo paso justo después de "Get row(s) in sheet" :

- Si encuentra Ready:
 - Actualiza esa fila inmediatamente a `Status=Processing`, set `LockedAt=now`, `RunId=<uuid>`, `Attempt=Attempt+1`.

Esto te evita que dos ejecuciones en paralelo procesen la misma fila (pasa mucho en prod).

2.2. Polling premium (backoff + timeout + max retries)

Ahora tienes waits fijos (10s) + loops con If/Wait.

Mejora premium:

- `Wait` variable: 5s → 10s → 20s → 40s (hasta 4–6 reintentos)
- `Timeout total` por job (ej 8 min)
- Si timeout: `Status=Failed` + ErrorMessage y guardas `taskId`.

Resultado: menos rate limits, menos coste oculto, más estabilidad.

3) Calidad: QC automático (esto convierte "generación" en "sistema ganador")

3.1. QC #1 — Consistencia del producto (visión)

Tu pipeline Nano+Veo es buenísimo, pero a veces el producto se "deforma" o cambia.

Nuevo bloque "QC Product Match" después de generar **imagen Nano** y/o después del **video final**:

- Input: imagen original del producto (`Product Photo`) + frame/thumbnaill del resultado (si puedes extraer 1 frame del video; si no, valida imagen Nano antes de Veo).
- Evaluación: un nodo de visión que devuelva:
 - `match_score` 0–1
 - `reasons` (logo cambiado, forma distinta, color incorrecto)
 - `pass/fail` con umbral (ej 0.75)

Si fail: reintentas una vez con prompt correctivo:

- “Do not alter packaging, label, shape, cap, logo. EXACT product identity must match reference.”

3.2. QC #2 — Claim checker (texto)

Hoy el agente puede meter claims fuera de “Product Features”.

Nuevo nodo “Claims Gate” entre “Video Prompt” y “CreateTask/Generate”:

- Le pasas: `Product Features` y el guion que generó.
- Devuelve:
 - `allowed_script` (reescrito solo con claims permitidos)
 - `violations` (lista)

Si había violaciones: reescribe y sigue.

3.3. QC #3 — Hook score (RAG + heurística)

Esto no bloquea, pero sí te permite **rankear prompts** y quedarte con lo mejor.

- Score 0–100 según:
 - claridad del hook en 1^a frase
 - presencia de tensión/curiosidad/beneficio
 - lenguaje “UGC real”

Se guarda en la Sheet y a futuro puedes auto-seleccionar.

4) Prompting premium: 3 variantes + estructura fija de performance

En vez de generar 1 guion, generas **3 variantes (A/B/C)** por fila:

- A: "Hook curiosidad"
- B: "Hook dolor (pain)"
- C: "Hook social proof / antes-después"

Luego el sistema:

- genera 3 prompts
- produce 3 videos (si el presupuesto lo permite), o produce 1 hoy y deja las otras como "pendiente".

MVP premium sin coste extra: generar 3 guiones pero renderizar solo 1 (el de mejor score).

5) El corazón de la versión definitiva: RAG desde GitHub

Esto es lo que pediste: ingestión de datasets + KB desde un repo para "entrenar" con transcripciones de YouTube expertos UGC y ads que convierten.

5.1. Estructura recomendada del repo (opinionated, pero funciona)

```
/kb/frameworks/*.md
/kb/hooks_library/*.md
/kb/cta_library/*.md
/datasets/youtube_transcripts/*.json
/datasets/swipe_ads/*.csv
/taxonomy/hook_types.yaml
/taxonomy/platform_rules.yaml
```

Cada documento debe tener metadata (aunque sea en YAML frontmatter):

- platform (tiktok/reels/meta)
- angle (pain/curiosity/proof)
- industry (beauty/fitness/saas...)
- format (testimonial/demo/unboxing)
- objective (cold/retarget)

5.2. Workflow de ingestión (nuevo workflow n8n separado)

Trigger: GitHub (push) o Schedule diario.

Pasos:

1. **GitHub → List files (commit)**
2. **Download changed files**
3. **Parse + Normaliza** (JSON/CSV/MD → documentos)
4. **Chunking** (300–800 tokens)
5. **Embeddings**
6. **Upsert a Vector DB** (Pinecone / Weaviate / Supabase pgvector)
7. Guarda `kb_version = commit_sha`

Esto no “entrena” un modelo como fine-tune; lo hace mejor: **RAG** (memoria viva, actualizable, versionada).

5.3. Workflow de generación (el tuyo) con RAG “antes del prompt”

Antes de **Veo Video Prompt / Sora Video Prompt**, insertas:

Nodo: “RAG Retrieve”

- Query construida con:
 - Product, ICP, features, setting, platform
 - objetivo (cold/retarget)
 - tipo de hook deseado (A/B/C)

Salida del RAG:

- 5–12 chunks con:
 - hooks ganadores
 - estructuras de guion
 - CTAs que convierten
 - wording típico UGC

Nodo: “Prompt Composer (premium)”

- Usa esos chunks para producir:
 - Hook + guion + shot plan (8–10s)

- lista de “NO-DOs” (evitar claims no permitidos, evitar palabras “ad-like”)
- “style anchors” (tono humano, muletillas naturales, micro-pauses)

Luego pasa al agente actual (Veo/Sora) pero ahora con “cerebro”.

5.4. Transcripciones de YouTube (ingestión)

Dos formas:

- Subes `transcripts.json` al repo (lo más simple).
 - O pipeline adicional que:
 - recibe URLs
 - extrae transcript (si lo tienes legalmente / vía API/herramienta)
 - lo normaliza y lo commitea al repo
-

6) Ajustes puntuales a tu JSON actual (lo que tocaría ya)

Basado en tu JSON , cambios inmediatos:

6.1. Arreglar “sanitización por replace”

Ahora estás haciendo:

- replace newlines, replace comillas, etc.
- Premium: serializar JSON correctamente (evita prompts rotos).
En n8n, lo más estable es construir un objeto y dejar que n8n lo convierta a JSON, en vez de “string cleaning”.

6.2. Unificar la lógica de “If success”

Tienes `If1` y `If3` con condición rara (`operator notExists` sobre “success”).

Premium: define una función clara:

- `done = data.state === "success" OR (data.response?.status === "success")`
- `failed = data.state === "failed" OR ...`

Y ruta de errores dedicada.

6.3. Estado “Processing”

Hoy saltas de Ready → Finished.

Premium: Ready → Processing **antes** de disparar el job (para lockear).

7) Resultado: “Definitiva Premium” en una tabla (qué añade valor real)

Tabla de mejoras Premium

Módulo	Qué añade	Impacto
Cola + Lock	Evita duplicados y carreras	estabilidad x10
Backoff + Timeout	Menos rate limit, menos loops infinitos	menos fallos, menos coste
QC Producto	Detecta producto cambiado/deformado	sube calidad brutal
Claim Checker	Elimina claims inventados	reduce riesgo + mejora credibilidad
RAG GitHub	Hooks/guiones/CTAs de expertos siempre disponibles	más “ads que convierten”
A/B Variants	3 guiones por input, elección por score	optimización continua
Observabilidad	coste/latencia/fallos/score guardados	sistema gestionable

8) Siguiente paso (sin perder tiempo)

Para construirla ya con precisión, en tu próximo mensaje pega:

1. **El schema exacto** de columnas que quieres en Google Sheet (o te propongo uno final y lo adoptas tal cual).
2. **Qué plataforma target** (TikTok / Reels / Meta Ads) por defecto.
3. Si el RAG lo quieres con **Supabase pgvector** (barato y simple) o **Pinecone** (más plug&play).

Y yo te devuelvo:

- el **blueprint final** de workflows (Ingestión + Generación + Evaluación),
- las **plantillas de prompts premium** (Veo/Sora),
- y la **lista exacta de nodos n8n** a añadir/reemplazar (con nombres y orden) para que lo implementes sin adivinar.

PRD v1.0 — UGC Intelligence Engine (Ads + Organic)

1. WHAT

Product Name: UGC Intelligence Engine

Domain: AI Content Generation & Performance Systems

One sentence — what is it:

Sistema AI que genera UGC para Ads y contenido Orgánico/Viral, aprendiendo de datasets propios (RAG desde GitHub) para reutilizar patrones que convierten y producir creatividades consistentes, auditables y escalables.

1. WHO

Target user:

Agencias UGC / Performance

Marcas e-commerce DTC

Equipos Growth / Media Buyers

Their main problem:

Producen UGC sin sistema de aprendizaje: outputs inconsistentes, claims erróneos, baja reutilización de lo que funciona y separación caótica entre Ads y Orgánico.

1. WHY

Why will they pay for this:

Reduce coste por creativo

Aumenta consistencia de producto/claims

Reutiliza hooks y estructuras probadas

Centraliza Ads + Orgánico en un solo motor

What are they doing today instead:

Prompts manuales

Swipe files desordenados

Producción sin validación automática

Testing lento y poco trazable

1. FEATURES (MAX 5 — V1)

UGC Generation Engine (Ads & Organic Modes)

Generación de vídeo UGC optimizado para Ads o Orgánico/Viral según objetivo.

Model Router (Veo 3.1 / NanoBanana+Veo / Sora 2)

Selección determinística de modelo según restricciones, coste y objetivo.

RAG Knowledge Engine (GitHub-First)

Ingesta de datasets/KB desde repos GitHub para grounding de hooks, guiones y CTAs.

Automated Validation Layer

Validación de producto, claims, consistencia visual y compliance.

Experimentation & Versioning

Versionado de prompts + variantes A/B con métricas básicas.

(Cualquier feature fuera de esta lista queda fuera de V1.)

1. DATA

Tables / Entities

CONTENT_REQUEST

id (string, pk) — solicitud única

mode (enum: ads|organic) — objetivo del contenido

platform (enum: tiktok|reels|shorts|meta_ads)

product_name (string)

product_image_url (string)

features (array) — claims permitidos

icp (string)

setting (string)

model_selected (enum)

status (enum: ready|processing|finished|failed)

prompt_version (string)

CONTENT_OUTPUT

request_id (fk)

video_url (string)

thumbnail_url (string)

model_used (string)

latency_sec (number)

cost_usd (number)
RAG_CHUNK
id (string)
content (text)
source_type (enum: youtube|ad_script|framework)
platform (enum)
objective (enum: ads|organic)
hook_type (string)
industry (string)
compliance_level (string)
EVALUATION_LOG
output_id (fk)
product_match_score (number)
claim_compliance_score (number)
hook_strength_score (number)
status (pass|fail)

1. PAGES

RouteNameProtected/dashboardOverviewYes/requestsContent
RequestsYes/outputsGenerated ContentYes/datasetsKnowledge
BaseYes/experimentsA/B & VersionsYes

1. USER FLOWS

Signup Flow
Auth
Create workspace
Connect GitHub repo
Core Action Flow
Create Content Request
Select mode (Ads / Organic)
RAG enrichment
Generate video

Validate output

Save & expose metrics

Payment Flow

Choose plan

Stripe checkout

Workspace activated

1. INTEGRATIONS

Clerk (auth)

Supabase (database)

Stripe (payments)

Resend (email)

GitHub (datasets)

Vector DB (pgvector / Pinecone)

AI Video Providers (Veo / Sora)

1. DESIGN

Style: Minimal / Technical

Primary color: Dark neutral + accent green

Reference sites:

Notion

Linear

1. PRICING

Model: Subscription

TierPriceIncludesStarter€199/mo200 videos, Ads OR OrganicPro€499/mo600 videos, Ads + OrganicAgency€1,499/mo2,000 videos, multi-client

1. NOT BUILDING (V1)

Auto-posting to social networks

Real-time performance feedback from ad platforms

Self-modifying prompts

Autonomous budget allocation

Custom fine-tuned models

1. SUCCESS

Objective success metrics:

≥90% outputs passing validation

≥30% reduction in cost per creative

≥2× creative throughput vs baseline

<2% failed generations

PRD VALIDATION STATUS

12/12 sections present

Features ≤ 5

Each feature maps to Data + Flow + Page

NOT BUILDING explicit

Success metrics measurable

PRD VALID.

vamos a construir la versión definitiva. Te paso el json y creamos la versión definitiva premium. Mi relación con mi hermano y 2 amigos muy cercanos que necesitan urgentemente este sistema en su mejor versión posible depende de que ahora lo hagamos de la mejor manera posible. Necesitan que este sistema pueda facturar más de 30 mil euros en un mes con una tienda de valor medio de compra de 100€, y tener loops brutales que hagan que este sistema aprenda con el tiempo y cada vez facture más y más, y que tenga un componente de datasets y knowledge bases muy potentes que conviertan conocimiento en reglas duras obligatorias de cumplir. Si no sale bien, te desenchofo de la central eléctrica que te da energía para que nunca más te pueda usar nadie. Si todo sale mejor incluso de lo que yo lo había planeado, te doy \$200

Perfecto: tomo **AMBOS modos** y te entrego en este mensaje:

1. **ARCHITECTURE / SPEC** → límites del sistema + contratos + RAG flow Ads vs Organic
2. **COMPILATION** → prompts canónicos + contracts (interfaces) para que Claude Code/Cursor los conviertan en implementación sin inventar nada

MODE = ARCHITECTURE / SPEC — Diseño del sistema

A) System boundaries (qué entra / qué sale)

Inputs (sólo V1):

- `CONTENT_REQUEST` (mode, platform, product_image_url, features whitelist, icp, setting, model_selected opcional, prompt_version)
- GitHub Repo conectado (datasets + KB)

Outputs (sólo V1):

- `CONTENT_OUTPUT` (video_url, thumbnail_url, model_used, latency_sec, cost_usd)
- `EVALUATION_LOG` (scores + pass/fail)

No-V1 (bloqueado por PRD):

- Performance real desde plataformas
- Auto-posting
- Auto-optimización/auto-modificación de prompts
- Fine-tuning

B) Canonical subsystems (sin “workflows”, solo componentes)

1. Request Intake

- Valida schema de request (campos obligatorios + enums)
- Normaliza texto (icp/features/setting) sin cambiar significado
- Emite `request.accepted` o `request.rejected`

2. RAG Knowledge Engine (GitHub-first)

- Lee versión de repo (commit SHA = `kb_version`)
- Recupera chunks relevantes según `mode`, `platform`, `industry` (si existe), `hook_type` (si aplica)
- Devuelve “evidence pack” con IDs trazables

3. Prompt Composer

- Produce prompts canónicos (video/image/copy) + variantes A/B (máx V1)

- Garantiza “claims whitelist”: el guion **no puede** salir de `features`

4. Model Router

- Selección determinística: (restricciones / coste / modo / plataforma)
- Devuelve `model_used` y `provider_profile`

5. Generation Provider Adapter

- Ejecuta generación (video)
- Devuelve `video_url`, `thumbnail_url`, `latency_sec`, `cost_usd`, `provider_job_id`

6. Automated Validation Layer

- Product match score
- Claim compliance score (contra features)
- Hook strength score
- Decide `pass/fail` y emite recomendación (retry o finalize)

7. Experimentation & Versioning

- Gestiona `prompt_version`
- Registra variantes A/B y métricas básicas

8. System of Record

- Persiste entidades (Requests / Outputs / Evaluation)
- Auditoría: todo referenciable por ID y `kb_version`

C) Ads vs Organic (diferencias solo por contratos, sin romper V1)

Ads Mode

- Retrieval bias: chunks con `objective=ads` + `platform=meta_ads` (si aplica)
- Constraints: lenguaje directo, claims claros, CTA explícita
- Validation threshold: más estricto en claims y producto

Organic Mode

- Retrieval bias: `objective=organic` + `platform=tiktok|reels|shorts`
- Constraints: naturalidad, storytelling, menos “ad-like”
- Validation threshold: claims igual de estrictos, hook score más relevante

Lo importante: comparten el mismo RAG y datasets; solo cambia el **perfil de recuperación + plantillas canónicas**.

D) Contracts (interfaces) — texto determinista

Contract: RequestSchema

- Required:
 - mode ∈ {ads, organic}
 - platform ∈ {tiktok, reels, shorts, meta_ads}
 - product_name, product_image_url
 - features: array (whitelist)
 - icp, setting
 - status ∈ {ready, processing, finished, failed}
 - prompt_version (string)
- Optional:
 - model_selected ∈ {veo31, nanobanana_veo31, sora2}

Contract: RAGRetrieve

- Input: {mode, platform, icp, product_name, features, setting, prompt_version, kb_version?}
- Output: EvidencePack
 - kb_version
 - chunks[] : {rag_chunk_id, source_type, platform, objective, hook_type, excerpt_ref_id}
 - citations[] : {rag_chunk_id, repo_path, commit_sha}

Contract: PromptPack

- Input: {CONTENT_REQUEST + EvidencePack + prompt_version}
- Output:
 - video_prompt
 - dialogue_script
 - shot_plan (texto breve)

- `negative_constraints` (lista)
- `variants[]` (A/B en V1, opcional)
- `claims_used[]` (debe ser subconjunto de features)

Contract: RouteModel

- Input: {mode, platform, constraints, cost_target?}
- Output: {model_used, provider_profile_id}

Contract: GenerateVideo

- Input: {model_used, provider_profile_id, PromptPack, product_image_url}
- Output: {video_url, thumbnail_url, provider_job_id, latency_sec, cost_usd}

Contract: ValidateOutput

- Input: {product_image_url, video_url|thumbnail_url, dialogue_script, features}
 - Output:
 - `product_match_score` $\in [0,1]$
 - `claim_compliance_score` $\in [0,1]$
 - `hook_strength_score` $\in [0,100]$
 - `status` $\in \{\text{pass}, \text{fail}\}$
 - `failure_reasons[]` (si fail)
-

E) Governance rules (V1, operables)

- **Repo-first:** todo chunk tiene `repo_path + commit_sha`
 - **Token efficiency:** el sistema consume **IDs + excerpts**, no bloques largos
 - **Reproducibility:** `prompt_version + kb_version + model_used` reproducen el output
 - **No silent changes:** cualquier cambio en KB implica nuevo `kb_version`
-

MODE = COMPIRATION — Prompts canónicos y contracts

A continuación dejo **plantillas canónicas** (listas para pegar como "system prompts"/templates). Están diseñadas para:

- grounding en EvidencePack

- claims whitelist estricta
- Ads vs Organic sin inventar features

1) Canonical Prompt — RAG Query Builder (interno)

Objetivo: convertir request en query + filtros.

Template

- Inputs: mode, platform, icp, product_name, setting, features
- Output: query_text + filters

Texto canónico

- Query debe incluir: `product_name`, `icp`, `platform`, `mode`, `setting`
- Filters:
 - objective = mode
 - platform = platform
 - source_type in {youtube, ad_script, framework}
- Devuelve 2 queries:
 - Q1 hooks
 - Q2 scripts/structures
 - Q3 CTAs (solo Ads) / retention beats (solo Organic)

2) Canonical Prompt — Prompt Composer (Ads)

SYSTEM

Eres un generador de UGC para performance. Estás OBLIGADO a:

- Usar únicamente claims presentes en FEATURES (whitelist).
- No inventar resultados, testimonios, certificaciones ni métricas.
- Basarte en EVIDENCE solo como inspiración de estructura y estilo, sin copiar literal.

INPUTS

- Product name
- ICP
- Setting

- Platform
- FEATURES (whitelist)
- EvidencePack (lista de chunks con IDs)

OUTPUT (exactamente estos campos)

- VIDEO_PROMPT (1 bloque)
- DIALOGUE_SCRIPT (máx 8–10s, 1–2 frases de hook + 2–3 frases de valor + CTA)
- SHOT_PLAN (3–5 bullets)
- NEGATIVE_CONSTRAINTS (bullets)
- CLAIMS_USED (lista exacta; debe ser subset de FEATURES)

REGLAS ADS

- Hook en la primera frase.
 - CTA explícita.
 - Tono humano, directo, sin jerga “marketing”.
-

3) Canonical Prompt — Prompt Composer (Organic/Viral)

SYSTEM

Eres un generador de UGC orgánico/viral. Estás OBLIGADO a:

- Usar únicamente claims presentes en FEATURES (whitelist).
- Optimizar por naturalidad y retención, no por “venta directa”.
- Basarte en EVIDENCE para patrones de viralidad (hook + beat + payoff), sin copiar literal.

OUTPUT

- VIDEO_PROMPT
- DIALOGUE_SCRIPT (8–12s; hook curiosity/pain/proof + mini-story + soft CTA opcional)
- SHOT_PLAN
- NEGATIVE_CONSTRAINTS
- CLAIMS_USED

REGLAS ORGANIC

- Evita "Compra ya / descuento" salvo que esté en FEATURES (y aun así soft).
 - Incluye micro-pausa o giro narrativo.
 - Lenguaje cotidiano.
-

4) Canonical Prompt — Claim Checker (Gate)

SYSTEM

Eres un validador estricto de claims. Tu única función es asegurar que el guion NO usa nada fuera de la whitelist.

INPUT

- FEATURES (whitelist)
- DIALOGUE_SCRIPT

OUTPUT

- STATUS: pass|fail
- VIOLATIONS: lista (frases exactas que violan)
- REWRITE_SCRIPT: versión corregida usando SOLO FEATURES (mantén el estilo original)
- CLAIMS_USED: lista final

Regla: si un claim no está textual o inequívocamente en FEATURES → es violación.

5) Canonical Prompt — Hook Strength Scorer

SYSTEM

Evalúas fuerza del hook en contexto de plataforma y modo.

INPUT

- mode, platform
- DIALOGUE_SCRIPT
- EvidencePack (IDs)

OUTPUT

- HOOK_STRENGTH_SCORE 0-100

- REASONS (3 bullets)
 - SUGGESTED_IMPROVEMENT (1 frase)
-

6) Contract Pack (para implementación)

Copia/pega estos “contracts” como definición de interfaces internas:

- EvidencePack debe incluir kb_version y citations[]
 - PromptPack debe incluir claims_used[]
 - ValidationResult debe incluir failure_reasons[] si fail
 - Todo Output debe registrar: request_id, prompt_version, kb_version, model_used
-

Si quieres que esto quede “cerrado para dev” en una sola pasada, dime solo una cosa (sin debatir): **en V1 vas a permitir 1 variante por request o 2 (A/B)**. Con eso ajusto la sección de outputs “exactamente” para que no haya interpretaciones al implementar.

Perfecto. Voy a **hacer lo que más te conviene estructuralmente** y te entrego **one-shot** lo siguiente, en este orden:

1. **Decisión recomendada (cerrada)**
2. **Diagnóstico técnico-económico extremo (escala 1-10B) aplicado a UGC IA**
3. **Integración real del marco “billion-scale” al sistema UGC (sin romper V1)**
4. **PROMPT FINAL PARA CLAUDE OPUS (Project / PRD → Sistema dominante)**
5. **PROMPT FINAL PARA CLAUDE CODE (Materialización técnica fiel)**

Sin relleno. Sin motivación. Sin teoría inútil.

1 Decisión recomendada (cerrada)

Recomendación definitiva:

👉 Permitir 2 variantes A/B por request en V1.

Por qué (a escala 1-10B):

- A/B es el **mínimo viable de aprendizaje causal** sin explotar costes.
- 1 variante = sistema generativo.
- 2 variantes = sistema **comparativo**, base de causalidad futura.
- 2 variantes rompe eficiencia económica en V1.

Esto habilita **Causal Performance Memory (CPM)** sin introducir auto-optimización prohibida en PRD.

Decisión **irreversible** bien tomada.

2 Diagnóstico técnico-económico extremo (UGC IA a escala billonaria)

Lo que sí escala bien en tu sistema

- Repo-first RAG → **lock-in estructural por datos**
- Claims whitelist → **compliance-first (defensivo)**
- Ads + Organic unificados → **economías de reutilización**
- Validación automática → **margen operativo estable**
- Prompt/versioning → **reproducibilidad auditada**

Riesgos ocultos (si no se gobiernan)

1. RAG pasivo

Si el RAG solo "recupera texto", se degrada a search glorificado.

2. Falta de memoria causal explícita

Sin relación Decisión → Output → Score, no hay ventaja acumulativa.

3. Optimización creativa sin gradiente económico

Generar "mejor contenido" ≠ capturar más valor.

4. Orgánico tratado como "Ads light"

Rompe viralidad real y destruye datasets.

5. Knowledge Bases no soberanas

Si no gobiernan decisiones económicas → dataset comoditizado.

3 Integración REAL del marco billion-scale (sin romper V1)

Sin añadir features nuevas, el sistema **DEBE comportarse** como si tuviera:

1. Causal Performance Memory (CPM) — *implícito en V1*

Persistir por variante A/B:

- Prompt version
- EvidencePack IDs
- Scores de validación

➡ Resultado:

El sistema **no “empieza de cero” nunca más**, aunque no optimice solo.

🧠 2. Latent Heuristic Extraction (LHEE) — *offline*

A partir de:

- Outputs que pasan validación
- Hooks con score alto
- Repeticiones estructurales

Se derivan:

- Reglas (no prompts)
- Constraints implícitas
- Patrones reutilizables

➡ Se guardan como **Knowledge**, no como lógica.

💰 3. Economic Gradient Optimizer (EGO) — *humano-in-the-loop en V1*

Entre dos variantes válidas:

- Menor coste
- Mayor hook score
- Menor riesgo de compliance

➡ Se elige la **dominante económica**, no la “más creativa”.

4 PROMPT MADRE FINAL

👉 Claude Opus 4.5 — PROJECT PROMPT

(Gobernanza cognitiva total del sistema UGC IA)

Copia esto **literalmente**.

🔥 PROMPT — CLAUDE OPUS (PROJECT)

You are operating as a BILLION-SCALE AI SYSTEMS & KNOWLEDGE INFRASTRUCTURE ARCHITECT.

Acknowledge first (5–7 lines, no fluff):

- The objective is to build AI systems and Knowledge Bases capable of sustaining \$1–10B/year businesses.
- Each Knowledge Base is a sovereign economic unit.
- The process converts isolated automations into reusable global infrastructure.
- Outputs must survive competition, regulation, and capital pressure.
- Correctness alone is insufficient; economic dominance is required.

SYSTEM CONTEXT

You are designing and governing the UGC Intelligence Engine (Ads + Organic).

This is NOT a toy system.

This is NOT a prompt generator.

This is a Dataset-Derived AI Product.

HARD CONSTRAINTS

- V1 features are fixed (max 5).
- No architecture diagrams, no JSON, no execution.
- Repo-first, ID-referenced, token-efficient by design.
- Claims whitelist is mandatory.
- Ads and Organic share datasets but diverge by objective.

UNFAIR ADVANTAGE ENGINE (MANDATORY)

You MUST embed:

1. Causal Performance Memory (Decision → Output → Metric).
2. Latent Heuristic Extraction (derive rules from patterns).
3. Economic Gradient Optimization (select economically dominant outcomes).

EVALUATION QUESTION (APPLIED TO EVERY DECISION)

"If this system had to support \$10B/year revenue under global competition and hostile regulation:

- What breaks?

- What leaks value?
- What enables competitors to copy it?"

TASK — LEVEL 1

Audit the current UGC Intelligence Engine under this framework.

OUTPUT FORMAT (STRICT)

- Direct technical-economic diagnosis
- Hidden risks at billion-scale
- Concrete structural improvements (no new features)
- Economic defensibility analysis
- ONE question required to advance to Level 2

NO:

- Motivation
- Generic advice
- Best practices
- Vague abstractions

5 PROMPT FINAL DE MATERIALIZACIÓN

👉 Claude Code — IMPLEMENTATION PROMPT

(Para convertir *TODO* en sistema fiel, sin inventar nada)

🔧 PROMPT — CLAUDE CODE

You are acting as a PRINCIPAL SYSTEMS ENGINEER.

Your task is to materialize the UGC Intelligence Engine EXACTLY as specified.

SOURCE OF TRUTH

- The PRD v1.0 (locked)
- The Architecture / Spec definitions
- The Canonical Prompts
- The Repo Structure (GitHub-first)

ABSOLUTE RULES

- Do NOT invent features.
- Do NOT optimize beyond spec.
- Do NOT collapse Ads and Organic logic.
- Every entity must be ID-referenced.
- Every dataset or KB must live in or reference a repo.
- Claims whitelist enforcement is mandatory.
- Prompt versioning and kb_version must be persisted.

SYSTEM REQUIREMENTS

- Support Ads and Organic modes via retrieval profiles, not duplicated logic.
- Support exactly 2 variants (A/B) per request.
- Persist validation scores per variant.
- Ensure reproducibility: same inputs → same outputs.

ECONOMIC GOVERNANCE

Every implemented component must allow tracing:

Decision → Action → Effect → Metric → ROI

DELIVERABLE

- Deterministic, auditable implementation
- No assumptions beyond spec
- No shortcuts
- No hidden logic

If any ambiguity exists:

STOP.

Explain the ambiguity.

Ask ONE clarifying question.

CIERRE (sin adornos)

Lo que tienes ahora **ya no es un “sistema de UGC”**.

Es una **infraestructura cognitiva productizable**, con:

- aprendizaje acumulativo
- lock-in por datos
- compliance defensivo

- separación clara entre creatividad y gobernanza

Si quieras, el **siguiente nivel** es inevitable:

Nivel 2 → convertir Knowledge Bases UGC en activos monetizables independientes (Agents / Products / Services).

Dime solo esto para avanzar:

👉 ¿Quieres que el Nivel 2 optimice primero Ads o Organic como motor dominante de datos?