IACADEMY

DE FUNDAMENTOS A ARQUITECTURA DE IA

MÓDULO 28

# IA para producto digital

Avanzado

iacedemy.com — 2026

MÓDULO 28

# IA para producto digital

**Nivel:** Avanzado

**Autor:** David Moya

**Publicación:** Mayo 2026

**Plataforma:** iacedemy.com

# Encontrar problemas que vale la pena resolver con IA

El 90% de los productos IA que fracasan lo hacen por la misma razon: resuelven un problema que a nadie le importa lo suficiente como para pagar. No empieces con "quiero hacer algo con IA". Empieza con "alguien esta pagando dinero por resolver esto manualmente".

Un buen problema para resolver con IA cumple tres criterios:

1. **Es repetitivo**: alguien lo hace muchas veces al dia/semana. La IA es buena automatizando tareas repetitivas.
2. **Requiere procesamiento de informacion**: leer, clasificar, resumir, extraer datos. Las tareas fisicas no son (todavia) para IA.
3. **Alguien ya esta pagando por resolverlo**: si nadie paga hoy (ni con dinero ni con tiempo), no pagaran manana.

> **Test rapido: merece la pena construir esto?**
>
> - Puedes describir el usuario en una frase? (no "todo el mundo")
> - El usuario hace esta tarea mas de 5 veces por semana?
> - Cada vez le toma mas de 10 minutos?
> - Pagaria 20 EUR/mes por eliminarlo?
>
> Si respondes "no" a cualquiera de estas preguntas, busca otro problema.

## Donde buscar problemas

Los mejores problemas estan donde ya trabajas. Tu sector, tu profesion, tus tareas diarias. Ejemplos reales de productos IA que funcionan:

- **Abogados**: resumir jurisprudencia de 200 paginas en 2. Producto: SaaS por suscripcion.
- **Inmobiliarias**: generar descripciones de propiedades a partir de fotos. Producto: API por uso.
- **Contables**: clasificar automaticamente facturas por categoria fiscal. Producto: integracion con software contable.
- **Soporte tecnico**: responder tickets nivel 1 automaticamente. Producto: chatbot entrenado con la base de conocimiento.
- **Recursos humanos**: filtrar CVs y generar shortlists. Producto: plugin para ATS existentes.

# MVP en semanas, no meses

Con las herramientas actuales, un MVP funcional de un producto IA se puede construir en 3-4 semanas. No necesitas un equipo de 10 personas. Necesitas una persona con Claude Code y un stack moderno.

## Semana 1: Validacion y prototipo

```
# Dia 1-2: Validar que la IA puede resolver el problema
# Prueba manual con Claude/GPT antes de escribir codigo

# Ejemplo: clasificador de facturas
prompt = """
Clasifica esta factura en una de estas categorias fiscales:
- Material de oficina (cuenta 629)
- Servicios profesionales (cuenta 623)
- Suministros (cuenta 628)
- Publicidad (cuenta 627)

Factura: "Factura de Google Ads por campana de marketing digital, 450 EUR + IVA"

Responde SOLO con la categoria y el numero de cuenta.
"""

# Si el LLM lo hace bien el 80%+ de las veces, tienes un producto viable
```

## Semana 2: Backend y API

```
# FastAPI + Supabase + LLM API
# Estructura minima del proyecto

proyecto-ia/
├── backend/
│   ├── main.py          # FastAPI app
│   ├── routes/
│   │   ├── classify.py  # Endpoint de clasificacion
│   │   └── auth.py      # Autenticacion via Supabase
│   ├── services/
│   │   ├── llm.py       # Wrapper del LLM (con cache)
│   │   └── storage.py   # Supabase client
│   └── requirements.txt
├── frontend/
│   └── ...              # Next.js (semana 3)
└── .env.example
```

```
# backend/main.py
from fastapi import FastAPI, Depends, HTTPException
from pydantic import BaseModel
import anthropic
import hashlib
import json

app = FastAPI()
client = anthropic.Anthropic()

# Cache simple para evitar llamadas repetidas al LLM
cache: dict[str, str] = {}

class ClassifyRequest(BaseModel):
    text: str
    categories: list[str]

class ClassifyResponse(BaseModel):
    category: str
    confidence: str
    reasoning: str

@app.post("/api/v1/classify", response_model=ClassifyResponse)
async def classify(request: ClassifyRequest):
    # Cache key basada en el contenido
    cache_key = hashlib.md5(
        f"{request.text}:{','.join(request.categories)}".encode()
    ).hexdigest()

    if cache_key in cache:
        return json.loads(cache[cache_key])

    response = client.messages.create(
        model="claude-sonnet-4-20250514",
        max_tokens=256,
        messages=[{
            "role": "user",
            "content": f"""Clasifica este texto en una de estas categorias: {request.categories

Texto: {request.text}

Responde en JSON: {{"category": "...", "confidence": "alta|media|baja", "reasoning": "..."}}"""
        }]
    )

    result = response.content[0].text
    cache[cache_key] = result
    return json.loads(result)
```

## Semana 3: Frontend y UX

Next.js con Supabase Auth. No reinventes la rueda de autenticacion. Usa los componentes de Supabase y enfocate en la experiencia del producto.

### Semana 4: Deploy y primeros usuarios

Deploy en Cloudflare Pages (frontend) y un VPS basico (backend). No necesitas Kubernetes. No necesitas microservicios. Un servidor con 4GB de RAM aguanta cientos de usuarios concurrentes con FastAPI.

## El stack para un producto IA

Este es el stack que recomendamos para un producto IA en 2026. Prioriza velocidad de desarrollo y coste minimo:

| CAPA | TECNOLOGIA | POR QUE | COSTE MENSUAL |
|---|---|---|---|
| Frontend | Next.js + Cloudflare Pages | Deploy gratis, edge, rapido | 0 EUR |
| Backend | FastAPI + VPS (Hetzner) | Async, rapido, Python (ecosistema IA) | ~5-15 EUR |
| Base de datos | Supabase (PostgreSQL + Auth) | RLS, auth integrada, tier gratuito | 0-25 EUR |
| LLM | Claude API / vLLM local | API para empezar, local para escalar | Variable |
| Pagos | Stripe | Estandar, webhooks fiables | 2.9% + 0.25 EUR/tx |
| Email | Brevo | Tier gratis 300 emails/dia | 0-25 EUR |
| Analytics | PostHog | Self-host o cloud EU, RGPD friendly | 0 EUR (tier gratis) |

**Coste total para arrancar: 5-40 EUR/mes.** Esto no es teoria. Es lo que cuesta realmente operar un producto IA funcional antes de tener miles de usuarios.

## Pricing de productos con IA

Pricing de productos con IA es diferente al SaaS tradicional porque tienes un coste variable significativo: cada llamada al LLM cuesta dinero. Hay tres modelos principales:

### Modelo 1: Suscripcion con limites

El usuario paga una cuota fija mensual y tiene un limite de uso. Ejemplo: 49 EUR/mes por 1000 clasificaciones. Es el modelo mas predecible para ti y para el cliente. **Recomendado para empezar.**

### Modelo 2: Pago por uso

El usuario paga por cada operacion. Ejemplo: 0.05 EUR por clasificacion. Barrera de entrada baja, pero ingresos impredecibles y dificil de comunicar.

### Modelo 3: Freemium

Tier gratuito limitado + tier de pago. Funciona bien para adquisicion, pero el tier gratuito te cuesta dinero (llamadas al LLM no son gratis).

> **Regla de oro del pricing con IA**
>
> Tu margen bruto debe ser minimo del 70%. Si tu coste de LLM por usuario es 10 EUR/mes, cobra minimo 33 EUR/mes. Si el margen es menor, tu negocio no escala. Calcula el coste por usuario activo antes de fijar precio.

```
# Calcular coste por usuario activo
def calcular_coste_usuario(
    llamadas_mes: int = 200,
    tokens_por_llamada: int = 500,
    coste_por_1k_tokens_input: float = 0.003,   # Claude Sonnet
    coste_por_1k_tokens_output: float = 0.015,
    tokens_output_por_llamada: int = 200
) -> dict:
    """Calcula el coste mensual de LLM por usuario activo."""
    coste_input = (llamadas_mes * tokens_por_llamada / 1000) * coste_por_1k_tokens_input
    coste_output = (llamadas_mes * tokens_output_por_llamada / 1000) * coste_por_1k_tokens_outp
    coste_total = coste_input + coste_output

    return {
        "coste_input_mes": round(coste_input, 2),
        "coste_output_mes": round(coste_output, 2),
        "coste_total_mes": round(coste_total, 2),
        "precio_minimo_70_margen": round(coste_total / 0.3, 2)
    }

# Ejemplo: usuario tipico con 200 llamadas/mes
resultado = calcular_coste_usuario()
print(resultado)
# {'coste_input_mes': 0.3, 'coste_output_mes': 0.6, 'coste_total_mes': 0.9, 'precio_minimo_70_m
```

## Go-to-market para productos IA

El go-to-market (GTM) de un producto IA tiene una regla fundamental: **habla del problema, no de la tecnologia**. A tu cliente no le importa que uses Claude o GPT-4. Le importa que clasifica sus 500 facturas al mes en 2 minutos en vez de 4 horas.

### Estrategia GTM en 4 pasos

1. **Demo publica**: crea una demo funcional sin registro. El usuario pega un texto y ve el resultado. Sin friccion. Esto genera el 80% de tus primeros leads.
2. **Contenido sobre el problema**: escribe articulos sobre el problema, no sobre tu producto. "Como clasificar 500 facturas al mes sin morir en el intento" funciona. "Nuestro clasificador IA revolucionario" no funciona.
3. **Pilots gratuitos**: ofrece 30 dias gratis a 5-10 empresas. No como "trial", sino como "pilot conjunto". Obtienes feedback real y testimonios.
4. **Escala con contenido + referidos**: cada cliente satisfecho te trae 2-3 mas. Facilita que compartan: caso de estudio conjunto, descuento por referido.

## Metricas que importan

Con un producto IA, necesitas monitorizar metricas estandar de SaaS mas metricas especificas de IA:

### Metricas SaaS clasicas

- **MRR (Monthly Recurring Revenue)**: ingresos recurrentes mensuales. La metrica norte.
- **Churn mensual**: porcentaje de clientes que cancelan. Objetivo: menos del 5%.
- **NPS (Net Promoter Score)**: encuesta simple de satisfaccion. Objetivo: mas de 40.
- **CAC (Customer Acquisition Cost)**: cuanto cuesta conseguir un cliente. Debe ser menor que 3x el LTV.

### Metricas especificas de IA

- **Coste por usuario activo**: cuanto gastas en LLM por cada usuario que usa el producto. Si sube, tu margen baja.
- **Precision percibida**: porcentaje de veces que el usuario acepta la respuesta de la IA sin corregir. Objetivo: mas del 85%.
- **Latencia p95**: tiempo de respuesta del percentil 95. Si supera 3 segundos, pierdes usuarios.
- **Tokens por request**: cuantos tokens consume cada peticion. Optimizar esto reduce costes directamente.

```
# Dashboard de metricas con PostHog
# Eventos a trackear desde el frontend

// Cuando el usuario hace una clasificacion
posthog.capture('classify_request', {
    category_count: categories.length,
    text_length: text.length
});

// Cuando acepta o rechaza el resultado
posthog.capture('classify_result', {
    accepted: true,  // o false si corrige
    latency_ms: responseTime,
    category: result.category
});

// Cuando convierte a paid
posthog.capture('subscription_started', {
    plan: 'pro',
    price: 49,
    trial_days_used: 14
});
```

## Escalar de 10 a 1000 usuarios

Los primeros 10 usuarios los consigues a mano: LinkedIn, demos, emails personalizados. De 10 a 100 funciona con contenido y referidos. De 100 a 1000 necesitas sistemas.

### De 10 a 100: el producto habla

En esta fase, el producto es tu mejor vendedor. Si el usuario usa el producto, le gusta y se lo cuenta a un colega, creces organicamente. Si necesitas "vender" activamente cada cuenta, el producto no es suficientemente bueno. Vuelve a mejorar la experiencia.

### De 100 a 1000: los sistemas hablan

Aqui necesitas: onboarding automatizado (sin llamada de ventas), self-service billing (Stripe), documentacion que resuelve el 80% de las dudas, y un pipeline de contenido constante (blog, YouTube, LinkedIn).

## Trampas tecnicas al escalar

Las trampas tecnicas de un producto IA son diferentes a las de un SaaS tradicional:

## Trampa 1: los costes de LLM escalan linealmente

Si un usuario te cuesta 1 EUR/mes en LLM, 1000 usuarios te cuestan 1000 EUR/mes. No hay economia de escala en las llamadas al LLM. Solucion: prompt caching agresivo, respuestas cacheadas para inputs frecuentes, modelos mas baratos para tareas simples.

```
# Estrategia de cache multi-nivel
import hashlib
import redis

r = redis.Redis()

async def classify_with_cache(text: str, categories: list[str]) -> dict:
    # Nivel 1: cache exacto (misma entrada = misma salida)
    exact_key = hashlib.md5(f"{text}:{categories}".encode()).hexdigest()
    cached = r.get(f"exact:{exact_key}")
    if cached:
        return json.loads(cached)

    # Nivel 2: cache semantico (entradas similares)
    # Usar embeddings para encontrar inputs similares ya procesados
    similar = find_similar_cached(text, threshold=0.95)
    if similar:
        return similar

    # Nivel 3: llamada al LLM (ultima opcion)
    result = await call_llm(text, categories)

    # Guardar en cache con TTL de 24h
    r.setex(f"exact:{exact_key}", 86400, json.dumps(result))
    store_embedding(text, result)

    return result
```

## Trampa 2: la latencia mata la experiencia

Las llamadas al LLM tardan 1-5 segundos. Con 100 usuarios concurrentes, el servidor puede saturarse si las llamadas son sincronas. Solucion: colas asincronas para tareas no interactivas, streaming para respuestas largas, rate limiting inteligente.

## Trampa 3: la precision baja cuando escalas

Con 10 usuarios, tu prompt funciona genial. Con 1000, aparecen edge cases que no previste. Solucion: monitoriza la precision percibida, crea un dataset de evaluacion que crece con cada error, itera el prompt semanalmente.

```
# Pipeline de evaluacion continua
# tests/evals/classify_eval.jsonl (crece con el tiempo)
{"input": "Factura Google Ads 450 EUR", "expected": "Publicidad (627)"}
{"input": "Recibo luz oficina marzo", "expected": "Suministros (628)"}
{"input": "Honorarios abogado constitucion SL", "expected": "Servicios profesionales (623)"}

# Ejecutar evals en CI antes de cada deploy
# python run_evals.py --threshold 0.85
# Si precision < 85%, el deploy falla
```

## Ejercicio practico

**Ejercicio M28: Disena tu producto IA**

1. **Identifica 3 problemas** en tu sector/trabajo que cumplan los tres criterios (repetitivo, procesamiento de informacion, alguien paga). Escribe una frase por cada uno.
2. **Valida con un prompt**: elige el problema mas prometedor y escribe un prompt en Claude/ChatGPT que lo resuelva. Prueba con 10 inputs reales. Anota la precision.
3. **Disena el pricing**: calcula el coste de LLM por usuario activo con el script de esta leccion. Define 3 tiers (free, pro, enterprise) con limites y precios que mantengan 70%+ de margen.
4. **Escribe el one-liner GTM**: una frase que describe el problema que resuelves (sin mencionar IA, LLM, ni tecnologia). Ejemplo: "Clasifica tus 500 facturas al mes en 2 minutos."
5. **Dibuja la arquitectura**: diagrama simple con las capas del stack (frontend, backend, DB, LLM, pagos). Identifica donde esta el cuello de botella si tienes 1000 usuarios.

**Bonus:** Crea un endpoint FastAPI funcional que resuelva el problema que elegiste. No necesitas frontend. Solo la API que acepta un input y devuelve el resultado.

# Conclusiones clave

### Key takeaways del M28

1. Un buen producto IA resuelve un problema que es repetitivo, requiere procesamiento de informacion y alguien ya paga por resolverlo manualmente.
2. Con Claude Code y el stack moderno (Next.js + FastAPI + Supabase), un MVP funcional se construye en 3-4 semanas. No necesitas equipo grande.
3. Pricing: suscripcion con limites es el modelo mas seguro para empezar. Margen bruto minimo del 70%. Calcula el coste por usuario activo antes de fijar precio.
4. GTM: demo publica sin registro, contenido sobre el problema (no sobre tu tecnologia), pilots gratuitos para conseguir testimonios reales.
5. Los costes de LLM escalan linealmente. Usa cache agresivo (exacto + semantico) para mantener margenes al crecer.
6. Monitoriza precision percibida, coste por usuario y latencia p95. Estas tres metricas de IA son tan importantes como MRR y churn.