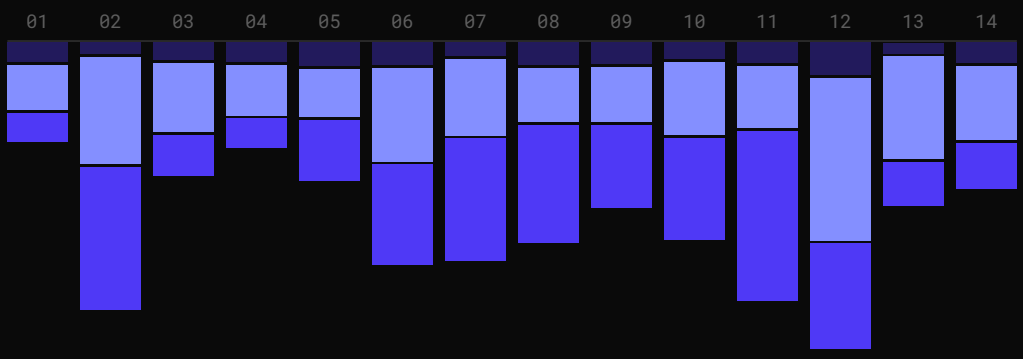
Helmcode

// GUÍA · 14 CAPÍTULOS · 3 PROFUNDIDADES DE LECTURA

Optimización de costes de IA

la guía práctica

Cómo reducir los costes de los LLM y de la infraestructura de IA sin sacrificar calidad.



LA GUÍA, MEDIDA

VISIÓN GENERAL · PRÁCTICA · EN PROFUNDIDAD

EDICIÓN DEL 4 NOV 2026

HELMCODE.COM/ES/GUIAS/OPTIMIZACION-COSTES-IA

// ÍNDICE

Índice

00 Introducción
Qué cubre la guía, cómo leerla y el caso Ledgerline.

PARTE I · ENTENDER LA ECONOMÍA

01 El coste real de una tarea de IA — 3 min
El precio por millón de tokens te dice menos de lo que crees.

02 Construye una línea base de costes útil — 7 min
Vincula cada llamada al modelo, cada cargo de herramienta y cada reintento a la funcionalidad y la tarea que los causaron.

PARTE II · ELEGIR MODELOS Y ENRUTAMIENTO

03 Elige el modelo que necesita la tarea — 3 min
Elige un modelo para cada operación poniendo precio a todo el camino hasta un resultado aceptado.

04 Enrutamiento, cascadas y presupuestos de razonamiento — 3 min
Envía cada petición al modelo que necesita, normalmente con una tabla de consulta y no con otra llamada a un modelo.

05 Qué cambian de verdad los modelos de pesos abiertos — 4 min
Elegir el modelo y elegir la infraestructura son dos decisiones distintas.

PARTE III · REDUCIR EL TRABAJO REPETIDO

06 Tokens, contexto y caché de prompts — 6 min
Recorta el contexto que una petición no usa y mantén un prefijo estable que la caché pueda reutilizar.

07 Memoria de agentes, caché de respuestas y trabajo duplicado — 6 min
Mantén pequeño el estado de trabajo de un agente y deja de pagar dos veces por el mismo resultado.

PARTE IV · GESTIONAR LOS FLUJOS DE INFERENCIA

08 Inferencia por lotes y asíncrona — 5 min
Si el resultado se necesita mañana, deja de pagar por defecto una respuesta en tres segundos.

09 Bucles de agentes, validación y reintentos — 4 min
Pon a los agentes límites de tiempo y de gasto, y ajusta cada política de reintentos al tipo de fallo.

10 Herramientas, búsqueda y RAG — 5 min
Envía al modelo los pasajes que necesita la respuesta, y cuenta lo que cuesta cada uno de más.

PARTE V · INFRAESTRUCTURA Y PRECIOS

11 Economía de la inferencia en GPU y del despliegue — 6 min
Dimensiona la capacidad de GPU para el tráfico pico y los objetivos de latencia antes de compararla con una API de pago por uso.

PARTE VI · LLEVARLO A LA PRÁCTICA

12 La economía de una funcionalidad de IA — 8 min
Una llamada de inferencia puede ser barata y la funcionalidad construida sobre ella puede seguir perdiendo dinero.

13 El manual de optimización — 4 min
Seis pasos para encontrar los pocos cambios que más van a bajar tu factura.

PARTE VII · NOTA DEL PROVEEDOR

14 Cuándo tiene sentido la inferencia de tarifa plana PROVEEDOR 4 min
Qué incluyen los planes de Helmcode, dónde terminan y cómo saber si uno de ellos bajaría tu factura.

REFERENCIA

A–Z Glosario
38 términos, cada uno enlazado a sus capítulos.

Introducción

Llamar a un LLM desde tu aplicación es fácil. Saber lo que te cuesta cada funcionalidad es más difícil.

Tu proveedor te informa de qué modelos has usado, cuántos tokens han procesado y cuánto has gastado. Si ese gasto era necesario, eso lo tienes que averiguar tú. Un agente de soporte puede buscar la misma información tres veces. Un pipeline de procesamiento de documentos puede extraer la misma factura después de cada reintento. Puede que estés usando un modelo de razonamiento grande para devolver una de cinco categorías.

Cada cosa parece barata por separado y se acumula al volumen de producción.

Algunas optimizaciones no merecen la pena. Un modelo más barato puede necesitar más reintentos. Un prompt más corto puede perder la evidencia necesaria para responder correctamente. Alojar los modelos tú mismo puede parecer atractivo hasta que cuentas las GPU ociosas y las personas que las operan.

Esta guía cubre las decisiones que cambian la economía de las aplicaciones de IA: elegir modelos, reducir el trabajo innecesario, gestionar el contexto, controlar los agentes, mejorar la recuperación y decidir cómo comprar u operar la inferencia. Cuando la respuesta depende de la carga de trabajo, explicamos lo que necesitas saber para decidir.

Para que los ejemplos estén conectados, el nivel Práctica de cada capítulo termina con la misma empresa ficticia. Ledgerline automatiza facturas de proveedores para empresas medianas y ejecuta tres operaciones de IA: extracción de facturas, un asistente de soporte y un agente de investigación que redacta informes de riesgo de proveedores. Empieza con 2.600 $ al mes, la mayoría de los capítulos cambian una línea de esa factura y las cifras se arrastran de un capítulo a otro.

QUÉ CUBRE LA GUÍA

Una factura de IA en producción rara vez se corresponde limpiamente con la funcionalidad que la generó. Analizamos el trabajo que hay detrás de esa factura, desde la inferencia y las herramientas hasta los reintentos, el contexto, la infraestructura y las suscripciones.

Los capítulos se agrupan según la decisión de ingeniería que estás tomando. Léelos en orden o ve directamente a un problema concreto.

CÓMO LEERLA

Cada capítulo tiene tres niveles de lectura. **Visión general** explica qué cambia y por qué. **Práctica** desarrolla los cálculos o los patrones de implementación. **En profundidad** cubre la mecánica y las contrapartidas allí donde importan.

Helmcode

// PARTE I

Entender la economía

01 El coste real de una tarea de IA

VISIÓN GENERAL
PRÁCTICA
EN PROFUNDIDAD

02 Construye una línea base de costes útil

VISIÓN GENERAL
PRÁCTICA
EN PROFUNDIDAD Define la aceptación antes de comparar costes de modelos
Un cálculo que une los eventos con los resultados

El coste real de una tarea de IA

El precio por millón de tokens te dice menos de lo que crees.

// NIVEL 1 Visión general

La mayoría de las comparaciones empiezan por los precios de los tokens de entrada y de salida. Esas cifras ponen precio a una llamada al modelo, pero los clientes pagan por un documento procesado, una incidencia resuelta o una tarea de programación terminada.

Una sola operación puede implicar una clasificación, una recuperación, dos llamadas al modelo, una búsqueda externa y un reintento. El precio de la respuesta final del modelo cubre solo uno de esos pasos. Una respuesta correcta de la API también puede ocultar una tarea fallida: un JSON sintácticamente válido puede contener el total equivocado de una factura.

Mide en su lugar el **coste por tarea aceptada**. Una tarea aceptada cumple los requisitos que tu producto fija para esa operación.

// NIVEL 2 Práctica

Empieza con:

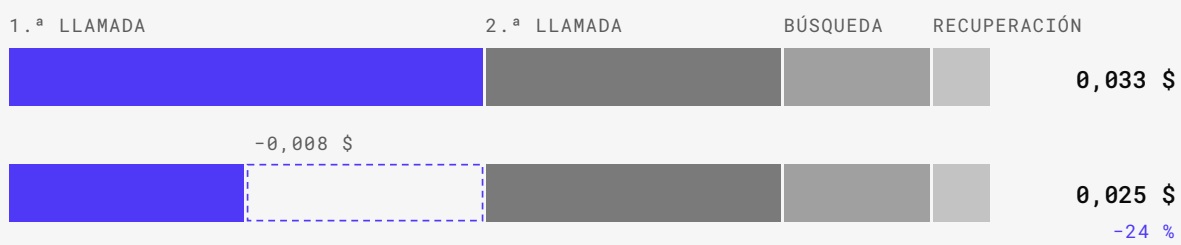
$$\text{Coste por tarea aceptada} = \frac{\text{Coste atribuible total del flujo}}{\text{Tareas aceptadas}}$$

Incluye los costes que apliquen: inferencia, embeddings y recuperación, herramientas externas, reintentos, cómputo, corrección humana y una parte de la capacidad fija. No cuentes dos veces una factura de plataforma compartida, y deja por escrito cómo la has repartido.

Un flujo de documentos hipotético:

EVENTO	COSTE
Primera llamada al modelo	0,016 $
Segunda llamada al modelo	0,010 $
Búsqueda	0,005 $
Recuperación	0,002 $
Total	**0,033 $**

Reducir a la mitad la primera llamada ahorra 0,008 $. El nuevo total es 0,025 $, alrededor de un 24 % menos, suponiendo que nada más cambia.



Una tarea documental, cuatro costes. Reducir a la mitad la primera llamada quita 0,008 $ a una tarea de 0,033 $, lo que abarata la tarea entera un 24 %.

Si ejecutas 10.000 tareas con un coste atribuible de 330 $ y aceptas 9.000, tu coste por tarea aceptada es de unos 0,0367 $. Una alternativa que cuesta 280 $ pero solo acepta 7.000 sale a 0,04 $ por tarea aceptada. La factura es menor, pero cada tarea aceptada cuesta más.

La hoja de cálculo tiene una pestaña que hace este cálculo para cada una de tus operaciones.

No necesitas una definición perfecta de calidad para empezar. Deja por escrito qué significa aceptar para cada operación.

> // CASO · LEDGERLINE.
>
> La factura del proveedor de Ledgerline en octubre fue de 2.600 $. Repartida entre 72.000 tareas aceptadas, sale a unos 0,036 $ por tarea, y esa media no dice casi nada. Una factura extraída cuesta 0,030 $, una respuesta de soporte 0,020 $ y un informe de riesgo de proveedor 0,40 $. Cada operación necesita su propia cifra antes de que nadie pueda decidir qué cambiar.

// NIVEL 3 En profundidad

Una llamada simple a la API sin caché cuesta:

$$C = \frac{\text{input_tokens} \times \text{input_price} + \text{output_tokens} \times \text{output_price}}{1.000.000}$$

Esa fórmula cambia cuando un proveedor distingue lecturas y escrituras en caché, tokens de razonamiento, tarifas de herramientas o modos por lotes. Si los tokens de razonamiento ya están incluidos en los tokens de salida que se reportan, no los cuentes dos veces.

En la infraestructura fija, separa los costes **marginales** de los **asignados**. Un servicio que cuesta 1.000 $ al mes procesa 50.000 tareas aceptadas a un coste asignado de 0,02 $ por tarea. Mientras quede capacidad libre, una tarea más que quepa en esa capacidad puede costar mucho menos que eso en el margen. Una vez que el servicio está saturado, la siguiente tarea puede requerir un nuevo nivel de capacidad.

Mira más allá de las medias. Unos pocos agentes de larga duración pueden disparar el coste por tarea del P95 muy por encima de la mediana, así que empieza a investigar por la cola.

Referencias: Modelos de razonamiento de OpenAI, Thinking en Gemini.

Construye una línea base de costes útil

Vincula cada llamada al modelo, cada cargo de herramienta y cada reintento a la funcionalidad y la tarea que los causaron.

// NIVEL 1 Visión general

Pongamos que un mismo modelo se encarga de la clasificación de soporte, la extracción de documentos y un agente de investigación interno. El panel del proveedor muestra que el modelo costó 2.600 $ el mes pasado, pero probablemente no qué funcionalidad lo gastó ni por qué.

Cada funcionalidad puede tener un problema distinto: prompts sobredimensionados en una, búsquedas de pago repetidas en otra, reintentos en la tercera. Antes de cambiar de modelo, atribuye el gasto a la operación responsable.

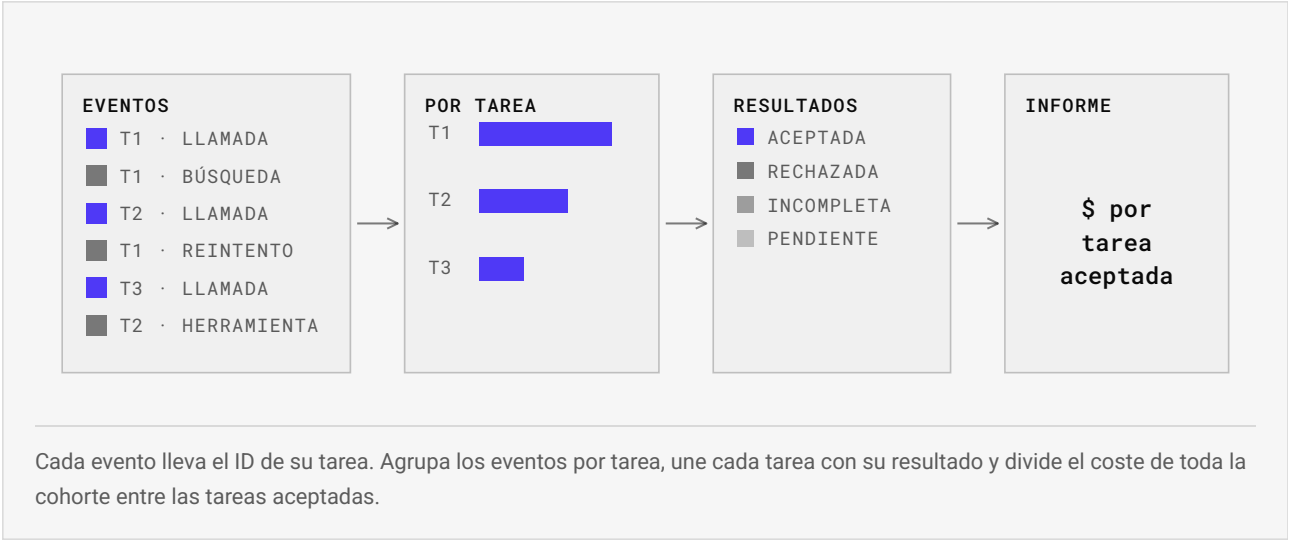
// NIVEL 2 Práctica

Asigna a cada operación un `task_id` y adjúntalo a cada evento del modelo, cada invocación de herramienta y cada reintento. Guarda el resultado final de la tarea por separado de los resultados de cada petición individual.

Un registro de evento mínimo necesita `task_id`, `operation`, `event_type`, `provider`, `model`, los recuentos de tokens de entrada, de salida y en caché cuando estén disponibles, el coste estimado o facturado, el número de intento y la latencia. Añade una marca de tiempo y la versión de la tabla de precios usada para calcular el coste. Para el análisis de costes suelen bastar los metadatos, así que no guardes prompts sensibles solo para construir un panel.

El informe de abajo cuenta tareas aceptadas, así que define *aceptada* antes de contar. En extracción, una tarea podría tener que pasar tres comprobaciones: la respuesta cumple el esquema (forma), las fechas, las monedas y los totales cumplen las reglas de negocio (coherencia) y los valores coinciden con el documento de origen

(exactitud). Cada operación necesita su propia definición, porque una respuesta de soporte o la ejecución de un agente tienen éxito de maneras distintas. Registra cada tarea como `accepted`, `rejected`, `incomplete` o `pending_review`, para que el trabajo fallido y el inacabado se queden en los datos con su coste.



Cada evento lleva el ID de su tarea. Agrupa los eventos por tarea, une cada tarea con su resultado y divide el coste de toda la cohorte entre las tareas aceptadas.

A partir de ahí, genera un informe breve:

OPERACIÓN	COSTE MENSUAL	TAREAS ACEPTADAS	COSTE POR TAREA ACEPTADA
Extracción	1.200 $	40.000	0,030 $
Soporte	600 $	30.000	0,020 $
Agente de investigación	800 $	2.000	0,400 $

DATOS ILUSTRATIVOS. El agente de investigación tiene el coste unitario más alto; la extracción tiene la factura absoluta más grande. Cuál optimizar depende del valor de la tarea, del trabajo innecesario y del ahorro factible.

En lugar de preguntarte qué modelo es caro, busca la operación que más paga en reintentos, la que reenvía una y otra vez contexto que no cambia, la que llama a más herramientas externas o la que muestra más distancia entre las ejecuciones normales y las costosas.

Lo que pagas se calcula a partir de los registros del proveedor, así que trátalos como la fuente de verdad. Obtén el uso y el coste de las API de uso y costes del proveedor o de las exportaciones de facturación, concílialos cada mes con los totales de tus eventos e investiga cualquier diferencia. La atribución es más fácil cuando el gasto se separa en origen: usa API keys distintas o etiquetas de metadatos por funcionalidad, equipo o entorno, para que el tráfico de pruebas no acabe en la factura de una funcionalidad de producción. Si ya usas una herramienta de observabilidad de LLM, o emites trazas que siguen las convenciones semánticas GenAI de OpenTelemetry (todavía en desarrollo), envía allí el mismo `task_id` en lugar de construir un segundo pipeline.

Después, asigna responsables a las cifras. Fija presupuestos y alertas por equipo. Usa showback, donde cada equipo ve su propio gasto, o chargeback, donde ese gasto se factura a su centro de coste. Mantén una previsión mensual sencilla por operación: el volumen esperado multiplicado por el coste actual por tarea aceptada. Cuando el gasto real se desvíe de ella, comprueba si ha cambiado el volumen o el coste unitario.

> // CASO · LEDGERLINE.
>
> La tabla de arriba es el primer informe de Ledgerline. Añadir un `task_id` a cada llamada, búsqueda y reintento llevó dos días. El resultado sorprendió al equipo: el agente de investigación produce menos del 3 % de las tareas aceptadas (2.000 de 72.000) pero el 31 % de la factura (800 $ de 2.600 $). La mayoría de los capítulos a partir de aquí cambian una de estas tres líneas.

// NIVEL 3 En profundidad

Una tabla de eventos sencilla en PostgreSQL puede sostener este análisis:

```sql
CREATE TABLE ai_cost_events (
  task_id TEXT NOT NULL,
  operation TEXT NOT NULL,
  event_type TEXT NOT NULL,
  model TEXT,
  input_tokens BIGINT DEFAULT 0,
  output_tokens BIGINT DEFAULT 0,
  cached_tokens BIGINT DEFAULT 0,
  cost_usd NUMERIC(18, 8) NOT NULL,
  attempt INTEGER DEFAULT 1,
  created_at TIMESTAMPTZ NOT NULL DEFAULT now()
);
```

Agrega primero los eventos **por tarea** y después únelos a una tabla de resultados de tareas. Así, una operación con ocho llamadas no cuenta como ocho tareas. Registra de forma explícita los costes fijos excluidos y los eventos incompletos, para que una factura incompleta nunca se presente como coste total.

Si la mezcla de tareas cambia de un mes a otro, el coste unitario medio puede moverse sin que cambie la eficiencia del modelo. Compara cosas comparables, o desglosa la métrica por tipo de tarea y complejidad.

Define la aceptación antes de comparar costes de modelos

La palabra *aceptada* necesita un significado operativo. Si no, el denominador se convierte en una cuestión de opinión, y la implementación más barata a menudo parecerá ganar porque descarta los casos difíciles.

Las tres comprobaciones de extracción necesitan herramientas distintas. Un validador de esquema se ocupa de la **forma**, y el código de la aplicación puede ocuparse de parte de la **coherencia**. La **exactitud** puede necesitar una comparación con el documento de origen o una revisión, porque una respuesta puede pasar las dos primeras comprobaciones y seguir siendo incorrecta.

En un asistente de documentación, una respuesta aceptada podría tener que responder a la pregunta, citar una fuente vigente y autorizada y evitar afirmaciones sin respaldo. En un agente de programación, la aceptación podría exigir que el cambio pedido esté presente, que no haya modificaciones ajenas y que se cumplan las condiciones de aceptación definidas. Una tarea de clasificación puede tener una etiqueta de referencia determinista; una tarea de investigación abierta puede necesitar una revisión graduada y estados explícitos de sin resolver.

Indica cuáles de los cuatro estados de tarea entran en tu denominador. En un informe mensual, define la cohorte y el corte: ¿tareas creadas este mes o tareas completadas este mes? Si mezclas las dos, el coste y las tasas de aceptación pueden no cuadrar solo porque algunas ejecuciones cruzan el cambio de mes.

Informa de la tasa de aceptación junto al coste por tarea aceptada. Cualquiera de las dos cifras por sí sola puede inducir a error:

OPERACIÓN	INICIADAS	ACEPTADAS	TASA DE ACEPTACIÓN	COSTE ATRIBUIBLE	COSTE / ACEPTADA
Extracción de facturas	10.000	9.600	96 %	300 $	0,03125 $
Síntesis de investigación	1.000	850	85 %	425 $	0,50 $

Escenario ilustrativo. Las dos operaciones producen tipos de resultado distintos, así que sus costes unitarios no se pueden clasificar directamente uno frente al otro.

Un cálculo que une los eventos con los resultados

Esta consulta de PostgreSQL hace explícito el denominador. Toma como cohorte las tareas **iniciadas** en el mes, de modo que el trabajo rechazado, incompleto y aún pendiente mantiene su coste en el numerador en lugar de desaparecer del informe. Supone una fila por tarea con su estado actual, y que la tabla de eventos incluye todos los costes variables directamente atribuibles. La capacidad fija compartida y la revisión humana necesitan sus propias asignaciones antes de que esto se convierta en un informe de TCO completo.

```sql
WITH task_cost AS (
  SELECT task_id, SUM(cost_usd) AS cost_usd
  FROM ai_cost_events
  GROUP BY task_id
), cohort AS (
  SELECT t.task_id, t.operation, t.status,
         COALESCE(c.cost_usd, 0) AS cost_usd
  FROM ai_tasks AS t
  LEFT JOIN task_cost AS c USING (task_id)
  WHERE t.created_at >= DATE '2026-10-01'
    AND t.created_at < DATE '2026-11-01'
)
SELECT operation,
       COUNT(*) AS started_tasks,
       COUNT(*) FILTER (WHERE status = 'accepted') AS accepted_tasks,
       COUNT(*) FILTER (WHERE status = 'pending_review') AS pending_tasks,
       SUM(cost_usd) AS variable_cost_usd,
       SUM(cost_usd) / NULLIF(
         COUNT(*) FILTER (WHERE status = 'accepted'), 0
       ) AS cost_per_accepted_task
FROM cohort
GROUP BY operation;
```

La consulta requiere una tabla `ai_tasks` con `task_id`, `operation`, `created_at` y `status` (`accepted`, `rejected`, `incomplete` o `pending_review`). Trata el mes como provisional mientras `pending_tasks` siga siendo significativo: cada tarea pendiente que acaba aceptada reduce el coste por tarea aceptada. En la contabilidad real, distingue los eventos de uso con coste cero o desconocido de los eventos confirmados como gratuitos. Concilia las sumas con las facturas, porque los cálculos de precios pueden pasar por alto descuentos por tokens en caché, cargos de herramientas o registros de uso que llegan con retraso.

Documenta la definición de aceptación y las exclusiones contables junto al informe, para que otro ingeniero pueda reproducir las cifras.

Las definiciones de las tablas y la consulta mensual también están en un archivo SQL que puedes descargar, con la tabla de tareas añadida.

Referencias: API de uso y costes de OpenAI, API de uso y costes de Anthropic, Spans de cliente GenAI de OpenTelemetry.

Helmcode

// PARTE II

Elegir modelos y enrutamiento

03 Elige el modelo que necesita la tarea
VISIÓN GENERAL
PRÁCTICA
EN PROFUNDIDAD

04 Enrutamiento, cascadas y presupuestos de razonamiento
VISIÓN GENERAL
PRÁCTICA
EN PROFUNDIDAD

05 Qué cambian de verdad los modelos de pesos abiertos
VISIÓN GENERAL
PRÁCTICA
EN PROFUNDIDAD El coste de cambiar más adelante

Elige el modelo que necesita la tarea

Elige un modelo para cada operación poniendo precio a todo el camino hasta un resultado aceptado.

// NIVEL 1 Visión general

Los equipos suelen elegir un modelo capaz para la primera funcionalidad y usarlo después en todas partes. Meses más tarde, ese modelo está clasificando tickets, resumiendo conversaciones, extrayendo campos e investigando incidentes de producción.

La configuración es fácil de entender, pero algunas operaciones acaban pagando por capacidades que no necesitan.

Un clasificador de tickets de cinco categorías y un agente que depura un fallo de pago intermitente son problemas distintos, y rara vez necesitan el mismo modelo y la misma configuración. Para cada operación, pon precio a todo el camino hasta un resultado aceptado, incluidas la validación, los reintentos y el escalado, antes de elegir un modelo por su precio por token.

// NIVEL 2 Práctica

Agrupa el trabajo según lo que requiere: clasificación acotada, extracción, resumen, preguntas y respuestas con fuentes, razonamiento sobre varios documentos y agentes que usan herramientas. Para cada grupo, registra los criterios de aceptación y las restricciones innegociables: latencia, tamaño del contexto, formato de salida, idioma, disponibilidad y tratamiento de los datos.

Después, haz una lista corta de modelos que puedan cumplir esos requisitos, incluidos candidatos de pesos abiertos cuando proceda. Una clasificación general de benchmarks no te dice si un modelo encaja con la tarea.

Para averiguarlo, construye un pequeño conjunto de evaluación con casos reales de esa operación, que incluya tareas que se aceptaron y tareas que se rechazaron. Ejecuta cada candidato contra él antes de cambiar, y otra vez después de cualquier cambio de prompt o de modelo. Esas ejecuciones cuestan tokens y tiempo de revisión, así que cuéntalas como parte de la decisión. Mira los resultados por tipo de caso y por modo de fallo además de la puntuación global, porque un modelo puede igualar la media y fallar en los casos que más importan.

Una comparación hipotética de dos enfoques para la misma operación de extracción:

	MODELO MÁS GRANDE	MODELO MÁS PEQUEÑO + ESCALADO
Llamada inicial	0,020 $	0,006 $
Validación (mismas comprobaciones en ambos)	0,001 $	0,001 $
Escalado	Ninguno	15 % × 0,020 $
Coste modelado por petición	0,021 $	0,010 $

Las dos columnas incluyen validación porque cualquiera de los dos modelos puede devolver una respuesta incorrecta. La comparación también supone tasas de aceptación iguales; si difieren, compara el coste por tarea aceptada. Que la columna más barata se sostenga depende de con qué fiabilidad la comprobación detecta los fallos, algo que trata Enrutamiento, cascadas y presupuestos de razonamiento.

Antes de añadir otro modelo, comprueba si el código normal puede hacer el trabajo. Dar formato a campos conocidos, comparar marcas de tiempo y la aritmética no suelen necesitar inferencia generativa.

> // CASO · LEDGERLINE.
>
> Ledgerline construyó un conjunto de evaluación de 300 facturas reales, 200 aceptadas y 100 rechazadas anteriormente, y ejecutó dos candidatos contra él. Cada ejecución del conjunto cuesta unos 9 $. Un modelo más pequeño con un 15 % de escalado al grande igualó la aceptación del modelo grande en el conjunto y dejó la extracción en 0,018 $ por factura aceptada: 40.000 × 0,018 $ = 720 $ al mes, frente a 1.200 $.

// NIVEL 3 En profundidad

Compara el uso de tokens facturables además de los precios unitarios publicados. Dos modelos pueden tokenizar la misma entrada de forma distinta, necesitar distinto nivel de detalle en el prompt, producir salidas de distinta longitud y consumir presupuestos de razonamiento distintos.

Una ventana de contexto más grande no garantiza un mejor uso de un documento largo; la calidad de la recuperación y la estructura del prompt pueden importar más que el límite anunciado. Cada modelo adicional también añade mantenimiento: configuraciones de proveedor, variantes de prompt, comportamiento de respaldo y monitorización.

El fine-tuning o la destilación son otra palanca. En una tarea acotada y estable, como un esquema de extracción que rara vez cambia, un modelo más pequeño entrenado con tus salidas aceptadas o con las respuestas de un modelo más grande puede sustituir a uno grande y generalista. Trae sus propios costes: preparar los datos de entrenamiento, las ejecuciones de entrenamiento, la evaluación, el alojamiento y el reentrenamiento cuando cambian las entradas o los requisitos. Las licencias de los modelos implicados también deben permitirlo, algo que trata Qué cambian de verdad los modelos de pesos abiertos. Reparte esos costes entre el volumen que esperas antes del siguiente reentrenamiento y súmalos a la comparación.

Mantén pequeño el número de modelos, y añade uno para una carga de trabajo distinta solo cuando el beneficio justifique ese mantenimiento.

Referencias: Licencia y destilación de DeepSeek-R1, Términos de uso de Gemma (derivados del modelo).

Enrutamiento, cascadas y presupuestos de razonamiento

Envía cada petición al modelo que necesita, normalmente con una tabla de consulta y no con otra llamada a un modelo.

// NIVEL 1 Visión general

El enrutamiento y las cascadas son técnicas relacionadas. El **enrutamiento** elige un modelo para una petición. **Una cascada** empieza con un modelo y escala a otro tras un fallo detectable o un resultado insuficiente.

Si tu aplicación tiene endpoints distintos para clasificación y para análisis complejo, probablemente ya sabe qué modelo necesita cada uno. Una consulta determinista es más sencilla que pedirle a un LLM que enrute a otro LLM.

El enrutamiento dinámico tiene sentido cuando un mismo endpoint recibe peticiones con requisitos muy distintos. Aun así, los dos tipos de error no cuestan lo mismo: una tarea fácil enviada a un modelo caro desperdicia dinero, mientras que una tarea difícil enviada a uno inadecuado puede hacer fallar toda la operación.

// NIVEL 2 Práctica

Empieza con políticas estáticas:

PYTHON

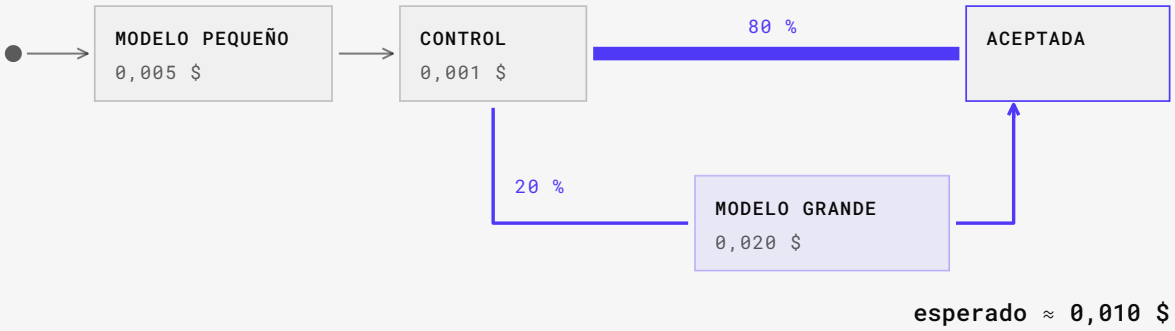
```python
MODEL_BY_OPERATION = {
    "classification": "small-model",
    "extraction": "extraction-model",
    "incident_analysis": "reasoning-model",
}

def choose_model(operation: str) -> str:
    return MODEL_BY_OPERATION.get(operation, "default-model")
```

Cuando una cascada está justificada, su coste esperado simplificado es:

$$C_{cascade} = C_{small} + p_{escalation} \times C_{large} + C_{validation} + C_{other_recovery}$$

Por ejemplo, una llamada inicial de 0,005 $, un 20 % de escalado a un modelo de 0,020 $ y 0,001 $ de validación dan un coste esperado de 0,010 $ antes de otro trabajo de recuperación. Esa cifra depende de tu señal de escalado y de tus requisitos de aceptación.



Todas las peticiones pagan el modelo pequeño y el control; solo el 20 % que no pasa el control paga además el grande. Coste esperado: unos 0,010 $.

Algunos modelos de razonamiento exponen controles de esfuerzo o de pensamiento, que pueden recortar la generación interna innecesaria en tareas acotadas sin cambiar de modelo. Los valores admitidos, los valores por defecto y la facturación varían según el modelo y el proveedor; no des por hecho que un ajuste genérico `none`, `low` o `high` funciona en todas partes.

Fija el esfuerzo de razonamiento por operación, junto a la elección del modelo. Un clasificador básico puede necesitar poco razonamiento extendido, mientras que una investigación de código difícil puede justificar más. No uses un límite bajo de tokens de salida como sustituto de un esfuerzo bajo: puede truncar la respuesta final después de que el razonamiento interno ya haya consumido el presupuesto.

> // CASO · LEDGERLINE.
>
> Seis de cada diez preguntas de soporte de Ledgerline preguntan en qué punto del flujo de aprobación está una factura. La aplicación ya lo sabe por su propia API, así que esas 18.000 preguntas pasan ahora por una consulta y un modelo pequeño con esfuerzo de razonamiento bajo, a 0,010 $ cada una. Las otras 12.000 mantienen la configuración actual a 0,020 $. Soporte baja de 600 $ a 18.000 × 0,010 $ + 12.000 × 0,020 $ = 420 $ al mes, sin modelo enrutador.

// NIVEL 3 En profundidad

El enrutamiento dinámico añade coste de clasificación y latencia. Si el enrutador cuesta 0,001 $ y evita de media 0,003 $ de inferencia posterior, la diferencia bruta es de 0,002 $ por petición, antes de contar las rutas equivocadas y el mantenimiento. A pequeña escala, puede que la arquitectura no compense.

Las cascadas dependen de una **señal de escalado útil**. Las comprobaciones de esquema y las violaciones de reglas de negocio son baratas de detectar; la corrección semántica es más difícil. Un modelo que declara una confianza alta puede seguir equivocándose, y pedir a otro modelo que juzgue cada respuesta puede borrar parte del ahorro.

Los controles de esfuerzo de razonamiento afectan a todo el flujo de trabajo. Con menos razonamiento, un agente puede hacer más llamadas a herramientas o repetir trabajo. En una API de pago por uso, menos tokens de razonamiento facturables pueden bajar el cargo de una llamada; en capacidad dedicada, el beneficio inmediato suele ser rendimiento (throughput) liberado.

Referencias: Esfuerzo de razonamiento de OpenAI, Esfuerzo de Anthropic, Niveles de pensamiento de Gemini.

Qué cambian de verdad los modelos de pesos abiertos

Elegir el modelo y elegir la infraestructura son dos decisiones distintas.

// NIVEL 1 Visión general

Cuando alguien propone sustituir un modelo propietario por Qwen, Gemma, DeepSeek o GLM, la conversación suele saltar directamente a las GPU. Pero elegir un modelo de pesos abiertos no significa que tengas que ejecutarlo tú.

Puedes usar una API de pago por uso, comprar capacidad dedicada a un proveedor u operar los pesos en tu propia infraestructura cuando la licencia lo permite. Esto importa para el coste, porque la organización que desarrolla el modelo no tiene por qué ser la que sirve cada petición.

Descargar los pesos no cuesta nada, pero cada una de esas opciones trae una factura de inferencia, pagada en tokens, en capacidad comprometida o en GPU. Un modelo de pesos abiertos tampoco es necesariamente “open source” en el sentido amplio. Publicar los pesos no implica publicar los datos de entrenamiento, un pipeline de entrenamiento reproducible ni derechos de uso sin restricciones.

// NIVEL 2 Práctica

Primero comprueba que el modelo concreto cumple los requisitos de la tarea y después compara formas de desplegarlo.

OPCIÓN	QUÉ PAGAS	PRINCIPAL CONTRAPARTIDA
API de pago por uso	Uso, a menudo tokens de entrada y de salida	Operación sencilla; la factura sigue al uso
Capacidad dedicada (gestionada)	Servicio o capacidad comprometidos	Coste más predecible; límites del plan y rate limits
Autoalojado	GPU, sistemas de soporte, operadores	Más control; capacidad ociosa y responsabilidad

Revisa la licencia por **versión** del modelo, no por familia: Qwen, Gemma, DeepSeek y GLM han publicado cada uno versiones con condiciones o dependencias distintas. Puede que la licencia del código de un repositorio no cubra los pesos, y las variantes destiladas pueden heredar condiciones de sus modelos base. Consulta la ficha del

modelo y el texto exacto de la licencia antes de desplegarlo comercialmente.

Unos pesos desarrollados en un país pueden ejecutarse en infraestructura operada en otro, así que el origen de un modelo dice poco sobre dónde se procesan tus datos. Para el tratamiento de datos, fíjate en quién opera el servicio, la región, el registro de logs, las herramientas externas y el contrato. Una API compatible con OpenAI no garantiza el mismo recorrido de los datos ni el mismo comportamiento de las funcionalidades entre proveedores.

> // CASO · LEDGERLINE.
>
> Para la extracción, la lista corta de Ledgerline incluía un modelo de pesos abiertos detrás de una API gestionada de pago por uso en la UE. Legal leyó la licencia de esa versión exacta, el equipo de seguridad comprobó dónde procesa los datos el proveedor, y el conjunto de evaluación del capítulo 3 decidió entre ese modelo y el modelo pequeño propietario. Nadie compró una GPU.

// NIVEL 3 En profundidad

El número de parámetros se traduce mal en coste de servicio; Economía de la inferencia en GPU y del despliegue trata el dimensionamiento de memoria para modelos densos y de mezcla de expertos.

Dos servicios que exponen la misma versión de los pesos pueden diferir igualmente en cuantización, tokenización o plantillas de chat, parsers de herramientas, configuración del razonamiento, rendimiento y disponibilidad. Cambiar una URL base puede reducir el trabajo de migración, pero sigues teniendo que validar y operar la nueva configuración.

El coste de cambiar más adelante

Los pesos abiertos y las API compatibles con OpenAI facilitan dejar un proveedor que sube precios, retira un modelo o cambia las condiciones. El cambio sigue costando trabajo:

- Vuelve a ejecutar tu conjunto de evaluación contra la nueva configuración. Elige el modelo que necesita la tarea explica cómo construir uno.
- Ajusta los prompts y los esquemas de herramientas cuando el nuevo servicio use otra plantilla de chat, otro parser de herramientas u otro soporte de salida estructurada.
- Vuelve a generar los embeddings de los documentos almacenados si cambia el modelo de embeddings, porque los vectores de modelos distintos no se pueden comparar entre sí.
- Exporta tus datos, logs y cualquier peso ajustado con fine-tuning, y presupuesta los cargos de salida de datos (egress) de la transferencia.
- Lee el contrato. Los compromisos mínimos, los créditos prepagados y los plazos de preaviso pueden encarecer la salida antes de empezar cualquier trabajo técnico.

Puedes reducir este coste desde el principio. Guarda los prompts, el conjunto de evaluación y el registro de peticiones en tus propios sistemas, en formatos que no dependan de un proveedor. Mantén una lista breve de las funcionalidades que usas y que solo ofrece un proveedor, como un modo de caché concreto o una interfaz de lotes, para que un plan de migración sepa qué tiene que sustituir.

Referencias oficiales de las versiones de los modelos: Qwen3, Condiciones de uso de Gemma, Ficha del modelo Gemma 4, DeepSeek-R1, DeepSeek-V3 y su licencia del modelo, Licencia de Qwen2.5, Licencia de GLM-4 9B, GLM-4.5, GLM-5. Cada versión tiene sus propias condiciones de licencia.

Helmcode

// PARTE III

Reducir el trabajo repetido

06 Tokens, contexto y caché de prompts

VISIÓN GENERAL
PRÁCTICA Imágenes, audio y documentos
EN PROFUNDIDAD Un límite de prompt estable en la práctica

07 Memoria de agentes, caché de respuestas y trabajo duplicado

VISIÓN GENERAL
PRÁCTICA
EN PROFUNDIDAD Cómo se acumula el crecimiento de la transcripción
Una operación, un resultado en vuelo

Tokens, contexto y caché de prompts

Recorta el contexto que una petición no usa y mantén un prefijo estable que la caché pueda reutilizar.

// NIVEL 1 Visión general

Los prompts crecen por motivos comprensibles: un ejemplo nuevo arregla un caso límite, otra instrucción mejora la consistencia y, en algún momento, la aplicación empieza a incluir la conversación entera para mantener la continuidad. Al final, cada petición lleva material que nadie sabe justificar con claridad.

Empieza por preguntarte por qué está ahí cada pieza de información. Una parte define la tarea, otra es evidencia y otra se repite porque la aplicación nunca aprendió a omitirla. Recortar el contexto irrelevante puede bajar el coste y dejar más claro el trabajo del modelo, siempre que se mantenga la evidencia de la que depende la respuesta.

La salida también cuenta. Si el producto necesita una categoría y una prioridad, un ensayo que explique la elección puede ser desperdicio. Especifica el formato exacto de salida, porque “sé conciso” es demasiado vago.

// NIVEL 2 Práctica

Inspecciona la **petición real** que se envía al modelo: instrucciones de sistema, ejemplos, definiciones de herramientas, pasajes recuperados e historial de la conversación, no solo el mensaje visible del usuario.

Para un asistente de soporte, puede bastar con la pregunta actual, la política aplicable y los datos relevantes de la cuenta, sin años de historial. Para la extracción, mejor un esquema explícito que explicaciones libres que el producto descarta. Conserva las excepciones, los identificadores originales y cualquier evidencia necesaria para que el resultado sea correcto.

Un ejemplo de coste con precios hipotéticos de 1 $ por millón de tokens de entrada y 4 $ por millón de tokens de salida:

	ANTES	DESPUÉS
Tokens de entrada	10.000	6.000
Tokens de salida	1.000	400
Coste por llamada	0,0140 $	0,0076 $

Es una reducción modelada del 45,7 % **para la llamada**, suponiendo que ambos resultados son igual de aceptables y que toda la entrada se cobra a precio sin caché.

La caché de prompts lo complica. Cuando muchas peticiones comparten un prefijo estable largo, un proveedor puede procesar y cobrar esa parte de otra forma, y un prompt más corto con datos dinámicos cerca del principio puede perder esa ventaja. Pon las instrucciones reutilizables y el material de referencia antes del contenido específico de cada petición cuando eso encaje con el comportamiento de caché del proveedor.

Imágenes, audio y documentos

La entrada que no es texto sigue sus propias reglas de facturación. Las imágenes suelen convertirse en tokens según su resolución, a menudo dividiéndolas en teselas de tamaño fijo, así que una imagen más grande cuesta más aunque el detalle extra no ayude a la tarea. El audio se cuenta normalmente por duración o se convierte en tokens. Los PDF y otros documentos se procesan a menudo página a página, a veces como texto extraído y como imagen renderizada de cada página a la vez.

Unos cuantos hábitos mantienen estas entradas bajo control:

- Envía la resolución que necesita la tarea. Reduce las capturas de pantalla y las fotos, y recorta la zona que importa, como una tabla o un campo de un formulario.
- Evita enviar la misma imagen o página en cada turno. El historial de la conversación arrastra los adjuntos, así que guarda el primer resultado (una descripción, los campos extraídos o una transcripción) y haz referencia a él en los turnos siguientes.
- Cuando la tarea solo necesita el texto, extráelo con una herramienta de OCR barata o un parser de documentos y envía ese texto al modelo.
- Comprueba cómo cuenta el proveedor cada modalidad antes de estimar. Los tokens por imagen o por página varían según el proveedor y el modelo, y una estimación basada solo en texto puede desviarse mucho.

> `// CASO · LEDGERLINE.`
>
> Cada petición de soporte de Ledgerline llevaba un bloque de 6.000 tokens de texto de políticas, con el nombre del cliente y una marca de tiempo al principio. Mover los campos específicos de la petición al final convirtió el bloque de políticas en un prefijo estable que el proveedor guarda en caché, y el formato de respuesta pasó de prosa libre a una respuesta estructurada breve. Soporte baja de 420 $ a 330 $ al mes.

// NIVEL 3

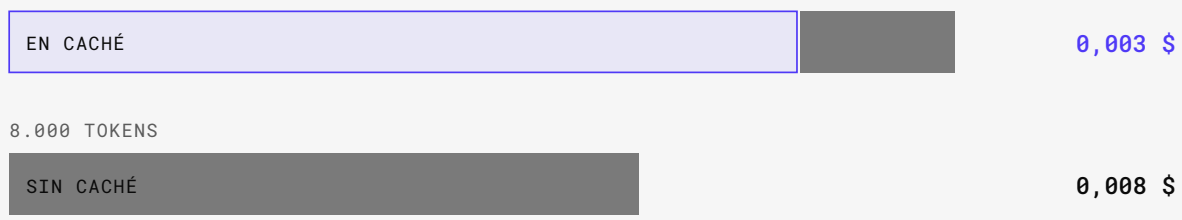
En profundidad

Hay tres mecanismos fáciles de confundir:

MECANISMO	REUTILIZA	QUÉ AHORRA
Caché de prompts del proveedor	Prefijo de entrada repetido	Cargos de entrada elegibles y/o procesamiento
Caché de prefijos del motor de servicio	KV cache almacenada de un prefijo compartido	Parte del cómputo de prefill
Caché de respuestas	Una salida anterior válida	Potencialmente la llamada de inferencia entera

Con la caché de prompts o de prefijos, el modelo en general sigue teniendo que generar una respuesta nueva. Ayuda sobre todo cuando entradas compartidas largas se repiten a menudo. No hay un descuento universal. Las escrituras y lecturas de caché, los TTL, los tamaños mínimos y la elegibilidad varían según el proveedor y el modelo.

A modo de ejemplo: 10.000 tokens de entrada en caché a 0,10 $/M más 2.000 sin caché a 1 $/M cuestan 0,003 $, mientras que un prompt reescrito de 8.000 tokens sin caché cuesta 0,008 $. En este escenario, el prompt más corto cuesta más.



Un prompt más largo puede costar menos cuando la mayor parte es un prefijo en caché. Tarifas hipotéticas: 0,10 $ por millón de tokens de entrada en caché y 1 $ por millón sin caché.

La caché de prefijos del motor de servicio funciona una capa por debajo. Cada petición construye una KV cache, el estado de atención de los tokens procesados hasta ese momento, y cada paso de decodificación la lee. La caché de prefijos, como la de vLLM, conserva la KV cache de un prefijo compartido y la reutiliza en peticiones posteriores que empiezan con los mismos tokens. Eso ahorra trabajo de *prefill* para el prefijo, pero el modelo sigue generando cada token de salida. En capacidad dedicada, el beneficio puede ser más rendimiento (throughput) y no una factura mensual menor.

Un límite de prompt estable en la práctica

El patrón simplificado en Python de abajo está pensado para una API que admite mensajes normales de sistema y de usuario. El material compartido se mantiene estable y el contenido específico de la petición va al final. El código **no** garantiza un acierto de caché, que depende del proveedor que sirve el modelo, del modelo, de la longitud mínima elegible y de las peticiones anteriores.

```python
SYSTEM = (
    "You answer questions about our public API. "
    "Use only the supplied reference material. "
    "If the source does not answer the question, say what is missing."
)

# Versioned and stable across requests until the documentation changes.
DOCS_VERSION = "api-docs-v17"
REFERENCE = load_documentation(DOCS_VERSION)

def messages_for(question: str) -> list[dict[str, str]]:
    return [
        {"role": "system", "content": SYSTEM},
        {"role": "user", "content": f"Reference ({DOCS_VERSION}):\n{REFERENCE}"},
        {"role": "user", "content": f"Question:\n{question}"},
    ]
```

En una aplicación real, `load_documentation` representa tu propio cargador de documentos versionado. No insertes una marca de tiempo ni un ID de petición aleatorio antes del material de referencia, porque cambia el principio de la petición cada vez. Mantén estables también las definiciones de herramientas y su orden cuando el proveedor las incluya en el prefijo cacheable.

Comprueba los campos de tokens en caché que devuelve la API antes de dar por hecho el ahorro. Cuando un servicio cobra por las escrituras o el almacenamiento en caché, calcula el coste efectivo del ciclo completo de reutilización en lugar de multiplicar el descuento de caché anunciado por todos los tokens de entrada. Un prefijo estable merece la pena incluso sin caché, porque hace que los cambios en el prompt sean deliberados y revisables.

Referencias: Caché de prompts de OpenAI, Caché de Gemini, Caché de prefijos de vLLM, Caché de prompts de Anthropic, Entradas de imagen de OpenAI, Recuento de tokens multimedia de Gemini, Soporte de PDF de Anthropic.

Memoria de agentes, caché de respuestas y trabajo duplicado

Mantén pequeño el estado de trabajo de un agente y deja de pagar dos veces por el mismo resultado.

// NIVEL 1 Visión general

Un agente de programación lee archivos, revisa logs y prueba enfoques que fallan. Al cabo de una hora, su transcripción puede contener miles de líneas de salida. El siguiente paso rara vez las necesita todas, solo el objetivo, las restricciones, los hallazgos confirmados, los enfoques fallidos y dónde está la evidencia. Ese conjunto más pequeño es el estado de trabajo del agente.

La compactación del contexto, la caché de respuestas y la idempotencia evitan cada una un tipo distinto de trabajo repetido, así que trátalas como problemas separados.

// NIVEL 2 Práctica

Mantén tres capas: la conversación reciente para la continuidad, un **estado de trabajo** duradero para las decisiones y las restricciones, y la evidencia guardada en otro sitio con referencias recuperables. Un registro compacto podría tener este aspecto:

JSON

```
{
  "objective": "Fix the webhook retry bug",
  "constraints": ["No duplicate charges", "Rollback required"],
  "findings": ["Worker may time out after downstream success"],
  "failed_approaches": ["Blind retry duplicates the action"],
  "evidence": ["src/webhooks.py", "incident-1042"],
  "next_step": "Check downstream idempotency support"
}
```

Guarda las restricciones exactas del usuario y el estado de negocio autoritativo fuera de los resúmenes en prosa, que pierden información. La compactación cuesta tokens y puede romper la reutilización del prefijo del prompt, así que compacta cuando la reducción esperada del contexto activo lo justifique, no automáticamente después de cada turno.

Para operaciones completadas que pueden producir con seguridad la misma respuesta, usa una **caché de respuestas exacta** con una clave formada por las entradas y versiones relevantes. La clave de una extracción de documentos podría incluir el hash del contenido, la versión del esquema, la versión del preprocesado, la configuración del modelo y el ámbito del tenant. Aplica la autorización por separado, porque una clave de caché no controla el acceso.

Cuando llegan dos peticiones idénticas al mismo tiempo, una caché de resultados terminados no basta, porque ambas pueden dar fallo de caché y lanzar la inferencia. Usa deduplicación en vuelo o un registro de operación compartido para que los llamantes esperen al mismo trabajo.

Mide el trabajo duplicado antes de construir una caché. Registra un hash de la petición en cada llamada al modelo, calculado a partir de las entradas que usaría una clave de caché. Cada día, informa por operación de la proporción de llamadas que repiten un hash anterior y de lo que cuestan esas llamadas. Cuenta aparte las peticiones idénticas que coincidieron en el tiempo, porque solo la deduplicación en vuelo las elimina.

ILUSTRATIVO: UNA OPERACIÓN DE EXTRACCIÓN HACE 40.000 LLAMADAS AL DÍA, Y 6.000 REPITEN UN HASH VISTO ANTES ESE MISMO DÍA: 6.000 / 40.000 = 15 %. A 0,004 $ POR LLAMADA, LAS REPETICIONES CUESTAN 6.000 × 0,004 $ = 24 $ AL DÍA, O 720 $ EN 30 DÍAS.

> // CASO · LEDGERLINE.
>
> Alrededor del 10 % de las facturas que llegan a Ledgerline son el mismo archivo subido dos veces, por el proveedor y por el cliente. Un hash del contenido más la versión del esquema y del modelo forman ahora la clave de una caché exacta, limitada a la cuenta del cliente, así que 4.000 de las 40.000 facturas mensuales se saltan la inferencia. La extracción pasa de 720 $ a 648 $.

// NIVEL 3 En profundidad

La caché de respuestas pregunta: “¿Tenemos un resultado todavía válido para estas entradas?”. La **idempotencia** pregunta: “¿Ya se ha enviado esta operación concreta que se pretendía hacer?”. Un agente que reintenta enviar un correo tras perder la respuesta de una herramienta no debe enviar dos correos solo porque el texto sea idéntico. Evitarlo exige una identidad de operación duradera.

Un lock distribuido tampoco evita los cargos duplicados. Si un worker se cae o agota el tiempo de espera después de enviar una petición, el proveedor puede ejecutarla igualmente, y el reintento la vuelve a ejecutar. Puedes garantizar que solo se confirma un resultado y aun así pagar dos llamadas al proveedor.

La caché semántica es más arriesgada que la caché exacta. “¿Puedo cancelar?” y “¿Puedo cancelar y que me devuelvan el dinero?” son preguntas parecidas que necesitan respuestas distintas. La similitud por sí sola no establece equivalencia, vigencia ni autorización. Empieza con reutilización exacta y ejecución deduplicada; añade reutilización semántica solo cuando puedas definir límites seguros.

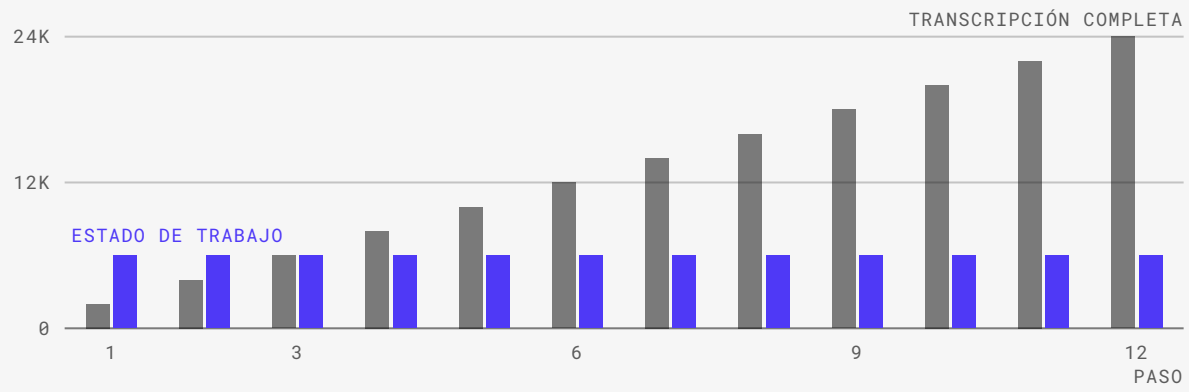
Cómo se acumula el crecimiento de la transcripción

Un agente que reenvía toda su transcripción vuelve a pagar en cada paso nuevo por todos los pasos anteriores. Si cada paso añade `t` tokens, el paso `k` envía unos `k × t` tokens de entrada, y una ejecución de `n` pasos envía `t × (1 + 2 + … + n) = t × n(n + 1) / 2`. La entrada total crece aproximadamente con el cuadrado del número de pasos. Un agente que envía un estado de trabajo acotado más el último paso envía más o menos lo mismo cada vez, así que su total crece de forma lineal.

ILUSTRATIVO: CADA PASO AÑADE 2.000 TOKENS. EL AGENTE COMPACTO ENVÍA UN ESTADO DE TRABAJO DE 4.000 TOKENS MÁS EL ÚLTIMO PASO DE 2.000 TOKENS, 6.000 TOKENS POR PASO.

PASOS (`N`)	TRANSCRIPCIÓN COMPLETA: 2.000 × N(N + 1) / 2	ESTADO COMPACTO: 6.000 × N
25	650.000	150.000
50	2.550.000	300.000
100	10.100.000	600.000

Duplicar la ejecución de 50 a 100 pasos multiplica la entrada de la transcripción completa aproximadamente por cuatro y la entrada compacta por dos. La columna compacta no incluye los tokens gastados en actualizar el estado de trabajo. La caché de prompts abarata el prefijo repetido (Tokens, contexto y caché de prompts), pero el número de tokens crece de la misma forma.



Tokens de entrada enviados en cada paso cuando cada paso añade 2.000 tokens. Reenviar la transcripción crece en cada paso; un estado de trabajo compacto se queda en unos 6.000.

Una operación, un resultado en vuelo

Una caché que comprueba `get(key)` y ejecuta la inferencia tras un fallo de caché permite que dos peticiones simultáneas lancen dos llamadas. La coordinación tiene que producirse **antes** de que empiece el trabajo caro.

Un registro de tarea duradero puede tener estos campos:

```text
(tenant_id, idempotency_key)  UNIQUE
request_hash                  required
status                        pending | running | completed | failed
result_reference              nullable
lease_owner, lease_expires_at  nullable
```

Al llegar la petición, crea o recupera el registro de forma atómica. Si la clave existe con un `request_hash` distinto, rechaza la discrepancia. Si está completada, devuelve su resultado; si está en ejecución, devuelve el identificador de la tarea o espera a que termine. Un lease permite una recuperación controlada cuando un worker desaparece. Publica el resultado final solo si el worker sigue siendo el dueño del lease; una transacción de base de datos o un fencing token pueden proteger esa actualización.

Esto protege el estado de la aplicación, pero **no** es una garantía de ejecución exactamente una vez para el proveedor de inferencia. Si el worker pierde su lease después de lanzar una petición remota, un worker de recuperación puede lanzar otra, así que registra ambos intentos si generan cargos. Para las herramientas que crean un pago, envían un correo o modifican un registro remoto, usa el contrato de idempotencia del sistema de destino siempre que exista.

Mantén esto separado de la caché de resultados terminados. Un hash del documento más la versión del esquema de extracción puede justificar reutilizar un resultado entre operaciones legítimas *distintas*, mientras que una clave de idempotencia identifica los reintentos de la *misma* operación pretendida. Mezclar ambas cosas provoca errores cuando un usuario pide a propósito un procesamiento nuevo.

Referencias: AWS: API idempotentes, Redis: locks distribuidos, Estado de la conversación de OpenAI.

Helmcode

// PARTE IV

Gestionar los flujos de inferencia

08 Inferencia por lotes y asíncrona

VISIÓN GENERAL
PRÁCTICA
EN PROFUNDIDAD Ejecutar trabajos por lotes del proveedor

09 Bucles de agentes, validación y reintentos

VISIÓN GENERAL
PRÁCTICA
EN PROFUNDIDAD Una ejecución con presupuesto, sin fingir que las estimaciones son facturas

10 Herramientas, búsqueda y RAG

VISIÓN GENERAL
PRÁCTICA
EN PROFUNDIDAD Costes de los embeddings

Inferencia por lotes y asíncrona

Si el resultado se necesita mañana, deja de pagar por defecto una respuesta en tres segundos.

// NIVEL 1 Visión general

Un pipeline de análisis de feedback clasifica miles de comentarios cada noche. Nadie lee el informe hasta la mañana siguiente, y aun así la aplicación puede estar usando el mismo modo de API en tiempo real que un asistente de cara al cliente.

Algunos proveedores aplican descuentos a la inferencia diferida, pero conviene separar tres cosas: la **ejecución asíncrona**, las **Batch APIs de los proveedores** y el **batching de inferencia a nivel de GPU**.

Un worker en segundo plano que llama a la API estándar es asíncrono para tu usuario, pero puede seguir pagando el precio estándar por token. La Batch API de un proveedor puede ofrecer una tarifa más baja para peticiones no urgentes. El batching continuo dentro de un motor de serving mejora el rendimiento (throughput) y no tiene nada que ver con los descuentos de la API.

// NIVEL 2 Práctica

Empieza por el plazo del producto. El soporte interactivo suele necesitar segundos; el procesamiento de documentos históricos puede tolerar horas. Los pasos dependientes de un agente limitan lo que puedes diferir, porque una acción posterior no se puede enviar hasta que el paso anterior produzca sus entradas.

Google, OpenAI y Anthropic documentan cada uno un modo por lotes con descuento para peticiones que pueden esperar, con resultados devueltos dentro de una ventana de entrega. En el momento de esta edición, Google indica una reducción de precio del 50 % para el uso elegible de la Gemini Batch API y un objetivo de entrega de 24 horas. Los descuentos, las ventanas y los modelos elegibles varían según el proveedor y cambian con el tiempo, así que revisa las condiciones vigentes de cada modelo y endpoint antes de contar con ellas.

Con un precio estándar hipotético de 0,012 $ por tarea, 500.000 tareas cuestan 6.000 $. Un modo por lotes elegible con una salida equivalente a mitad de precio cuesta 3.000 $ en cargos de inferencia. El ahorro neto puede quedarse por debajo de 3.000 $ cuando cuentas la gestión de los trabajos, las peticiones fallidas o caducadas, los reenvíos y cualquier trabajo que siga necesitando ejecución inmediata.

Haz el seguimiento de cada elemento del lote por separado, para poder reparar un trabajo que ha fallado en parte sin reenviar el lote entero. La sección En profundidad, más abajo, cubre los detalles.

> // CASO · LEDGERLINE.

> Una cuarta parte de las facturas de Ledgerline llega en importaciones masivas mensuales que nadie lee hasta la mañana siguiente. Ahora pasan por el modo por lotes del proveedor a mitad de precio, con cada factura seguida por su propio ID. Esa cuarta parte costaba 162 $ de los 648 $, así que la mitad de precio ahorra 81 $ y la extracción se queda en 567 $ al mes.

// NIVEL 3 En profundidad

En la inferencia autoalojada, el batching es una técnica de rendimiento: el motor de serving intercala peticiones de distinta longitud para que la GPU siga ocupada. La ganancia depende de la carga de trabajo, de la presión sobre la KV cache y del presupuesto de latencia, y Economía de la inferencia en GPU y del despliegue cubre la mecánica.

Un mejor rendimiento reduce el coste asignado a cada tarea, pero **la factura de capacidad sigue siendo la misma** salvo que el cambio te permita reducir o evitar capacidad.

El procesamiento por lotes también se combina con la caché y la deduplicación. Si solo cambió el 5 % de los documentos de ayer, saltarte los que no han cambiado puede ahorrar más que el descuento del precio por lotes.

Ejecutar trabajos por lotes del proveedor

Antes de enviar, escribe cada elemento en tu propio almacén con su ID de aplicación, un hash de la entrada, el modelo y la versión del prompt, el ID de trabajo del proveedor y un estado como `pending`, `submitted`, `succeeded`, `failed` o `expired`. Pon el ID de aplicación en el campo de identificador personalizado de la petición cuando la API lo ofrezca, y úsalo para cruzar los resultados con tus registros. Un fichero de resultados puede volver en un orden distinto al de la entrada, y pueden faltar algunos elementos.

Un trabajo terminado puede contener elementos correctos, elementos que devolvieron un error y elementos que caducaron antes de que el proveedor los procesara. Trata cada uno por separado. Una petición mal formada o una entrada que supera el límite de contexto volverá a fallar si no la cambias, así que corrígela o descártala. Los elementos caducados, los errores de rate limit (límite de tasa) y los errores transitorios del servidor sí merecen un reenvío.

Pon los elementos reintentables en un trabajo nuevo que conserve los ID de aplicación originales, y deja en paz los que han ido bien. Limita los intentos por elemento, por ejemplo a dos reenvíos, y después pasa lo que siga fallando a una cola de revisión. Sin límite, un fallo sistemático, como una plantilla de prompt que siempre supera el límite de contexto, sigue generando trabajos nuevos.

Haz que el procesamiento posterior sea idempotente. Un worker puede caerse después de leer un fichero de resultados y volver a leerlo, y un elemento que reenviaste tras un timeout puede seguir produciendo un resultado del primer intento. Escribe los resultados con un upsert cuya clave sea el ID de aplicación y la versión del prompt, y haz que los consumidores se salten los ID que ya han tratado. Un resultado duplicado cuesta entonces tokens, pero no crea un segundo ticket, correo o fila en la base de datos.

Cuenta todo lo que ha costado el trabajo. Incluye los cargos de los elementos cuya salida descartaste, los cargos de cada reenvío y cualquier elemento que pasaste a la API en tiempo real a precio estándar. Divide ese total entre el número de elementos que produjeron un resultado utilizable. Esa es la cifra que hay que comparar con

el procesamiento en tiempo real y, con una tasa de fallos alta, puede quedar muy por encima del precio por lotes anunciado.

Decide qué pasa con los elementos que no llegan al plazo antes de que se ejecute el primer trabajo. Si el informe de la mañana tiene que estar a las 08:00 y la ventana del proveedor permite hasta un día, un trabajo enviado a medianoche puede no terminar a tiempo. Las opciones son mandar los elementos tardíos a la API en tiempo real, publicar el informe con esos elementos marcados como pendientes u omitirlos cuando los datos vayan a estar desfasados de todos modos. Cada opción tiene un coste distinto, así que el responsable de producto debería elegir una y el pipeline debería implementarla de forma explícita.

Referencias: Gemini Batch API, Modos de optimización de Gemini, OpenAI Batch API, Procesamiento por lotes de Anthropic.

Bucles de agentes, validación y reintentos

Pon a los agentes límites de tiempo y de gasto, y ajusta cada política de reintentos al tipo de fallo.

// NIVEL 1 Visión general

La petición inicial de un agente no te dice cuánto va a costar. Puede inspeccionar un repositorio, llamar a varias herramientas, revisar hipótesis e intentar un arreglo. Algunas tareas necesitan ese trabajo. Otras se atascan repitiendo la misma búsqueda o reparando la misma respuesta no válida.

Un límite de cinco pasos para todos los agentes es demasiado tosco para controlar el coste. Diez llamadas cortas a un modelo pequeño cuestan mucho menos que diez llamadas largas a un modelo de razonamiento caro. Necesitas límites de tiempo y de gasto, y una forma de detectar el trabajo improductivo.

Los fallos también son distintos. Una sobrecarga temporal del proveedor, un JSON no válido y unos datos extraídos incorrectos son problemas diferentes, y reintentar los tres con el mismo prompt rara vez es la mejor política.

// NIVEL 2 Práctica

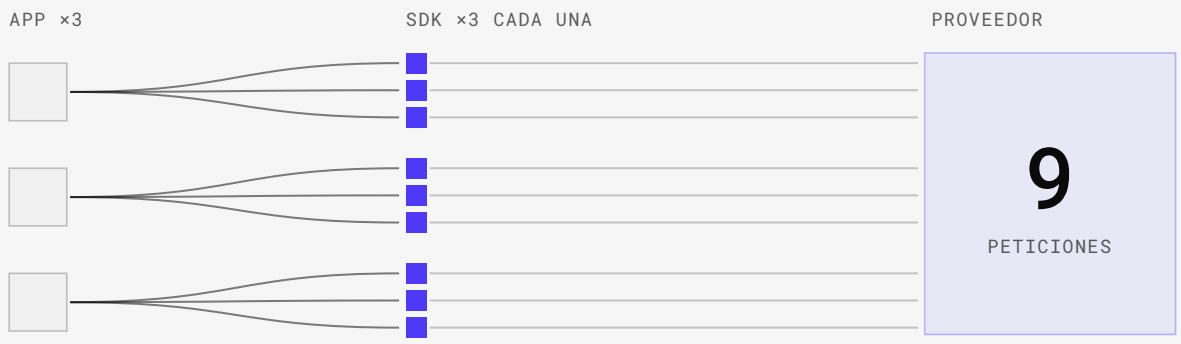
Presupuesta los agentes según la operación que realizan. Registra las llamadas al modelo, los cargos de herramientas externas, el tiempo real transcurrido y un total monetario estimado. Aplica los límites en el código de la aplicación, no pidiéndole al modelo que “gaste menos de 1 $”. Si el presupuesto tiene que ser estricto, reserva el coste estimado antes de lanzar trabajo concurrente y después concilia el uso real.

Define qué devuelve el agente cuando se detiene. Al llegar al límite del presupuesto, debería informar de los hallazgos confirmados, los cambios realizados, el trabajo pendiente y el motivo por el que se detuvo. Un resultado parcial que puedes rastrear vale más que un fallo genérico que tira el progreso a la basura.

Ajusta la política de recuperación al fallo:

FALLO	MEJOR PRIMERA RESPUESTA
Sobrecarga transitoria o error de red	Reintentos acotados, backoff y jitter
Esquema no válido	Forzar la salida estructurada soportada y validar de forma determinista
Falta de evidencia	Recuperar o aportar la fuente que falta
Interpretación semántica incorrecta	Reevaluar el modelo, la entrada y el método de verificación
Parámetro no soportado o crédito agotado	Corregir la configuración o la facturación, no reintentar a ciegas

Vigila la amplificación de reintentos. Si un SDK hace hasta tres intentos y tu aplicación lo envuelve en otros tres, el servicio de origen puede ver nueve peticiones. Fija un plazo global y conoce el comportamiento de reintentos de cada capa.



Tres intentos de la aplicación alrededor de un SDK que también reintenta tres veces pueden llegar al proveedor nueve veces por una sola tarea.

// CASO · LEDGERLINE.

El agente de investigación de Ledgerline promediaba 0,40 $ por informe aceptado, pero su ejecución P95 costaba varios dólares: una ejecución de cada veinte seguía buscando documentos regulatorios que no existían. Un tope de 1,00 $ por ejecución, una comprobación de búsquedas repetidas y un estado de parada que devuelve los hallazgos obtenidos hasta ese momento bajaron la línea de investigación de 800 $ a 640 $ al mes, con el mismo número de informes aceptados.

// NIVEL 3 En profundidad

La validación tiene su propio coste. Un parser de JSON es barato. Un segundo LLM que revise cada respuesta puede aumentar de forma considerable el gasto de inferencia y aun así pasar por alto errores semánticos. Usa comprobaciones deterministas siempre que puedas: restricciones de esquema, códigos de estado conocidos, aritmética, identificadores y datos de referencia fiables.

Para un modelo simple de reintentos acotados, en el que cada intento cuesta `C`, tiene éxito de forma independiente con probabilidad `p > 0` y se permiten `N` intentos:

$$\text{Intentos esperados} = \frac{1 - (1-p)^N}{p}$$

$$\text{Coste esperado} = C \times \text{Intentos esperados}$$

Con `p = 0,8` y `N = 3`, eso son 1,24 intentos, así que la llamada cuesta de media alrededor de un 24 % más que `C`. Con `p = 0,4`, son 1,96 intentos, y el 22 % de las tareas sigue fallando después del tercer intento.

Los reintentos reales rara vez se comportan de forma tan ordenada. Los errores del modelo suelen estar correlacionados entre intentos, los intentos pueden costar cantidades distintas y algunos fallos necesitan recuperación mediante herramientas o personas. En los agentes, guarda un registro de los enfoques fallidos para que el agente no los repita después de una compactación del contexto.

Mide el progreso por los cambios de estado, como evidencia nueva, hipótesis descartadas, trabajo completado o un enfoque distinto. Que el propio agente afirme que está avanzando no cuenta.

Una ejecución con presupuesto, sin fingir que las estimaciones son facturas

Mantén el presupuesto del agente fuera del modelo. El patrón mínimo de abajo hace una **reserva** antes de lanzar una llamada. Supone que un único worker actualiza el estado local; la concurrencia en producción necesita una reserva atómica en base de datos u otro mecanismo de sincronización.

```python
from dataclasses import dataclass
from decimal import Decimal

@dataclass
class RunBudget:
    ceiling: Decimal
    reserved: Decimal = Decimal("0")
    actual: Decimal = Decimal("0")
    model_calls: int = 0
    tool_calls: int = 0

    def reserve(self, predicted: Decimal) -> None:
        if self.actual + self.reserved + predicted > self.ceiling:
            raise RuntimeError("Task budget exhausted")
        self.reserved += predicted

    def reconcile(self, predicted: Decimal, charged: Decimal) -> None:
        self.reserved -= predicted
        self.actual += charged
```

La reserva solo decide si una llamada puede empezar; no pone un tope a la factura. El cargo real puede superar la estimación, los proveedores pueden facturar una petición que agotó el timeout y varios workers podrían entrar en condición de carrera salvo que la reserva sea atómica. Para limitar la exposición, acota los tokens y las herramientas donde se pueda, fija plazos, impide la concurrencia no autorizada y define qué pasa al llegar al tope.

Las llamadas de pago a búsqueda, navegación y procesamiento de documentos entran en el mismo presupuesto que las llamadas al modelo.

Referencias: AWS: backoff y jitter, Rate limits de OpenAI, Salidas estructuradas de OpenAI, Errores de la API de Anthropic y reintentos del SDK.

Herramientas, búsqueda y RAG

Envía al modelo los pasajes que necesita la respuesta, y cuenta lo que cuesta cada uno de más.

// NIVEL 1 Visión general

Un asistente de documentación recibe una pregunta, busca en una base de datos vectorial y envía veinte pasajes al modelo. Los dos primeros contienen la respuesta; el resto están relacionados pero sobran. La búsqueda puede ser barata, pero el modelo sigue procesando cada token devuelto.

Los costes de RAG abarcan la ingesta, los embeddings, el almacenamiento, la recuperación, el reranking, el contexto añadido al prompt y la respuesta final del modelo. La búsqueda web de pago, el OCR y otras herramientas pueden sumar cargos aparte. Optimizar solo la consulta vectorial o solo el modelo de generación pasa por alto cómo interactúan los dos.

// NIVEL 2 Práctica

Contabiliza por separado los costes en tiempo de consulta y en tiempo de ingesta. Una base de conocimiento casi estática que recibe millones de preguntas tiene una economía distinta a la de un archivo que se actualiza a menudo y recibe pocas consultas.

No trates `top_k`, el número de pasajes recuperados que se envían al modelo, como un mando universal de calidad. A 500 tokens por pasaje, veinte pasajes añaden unos 10.000 tokens de entrada; cinco añaden 2.500. A 2 $ por millón de tokens de entrada sin caché, la diferencia es de 0,015 $ por petición, o 1.500 $ en 100.000 peticiones. Pero si la respuesta depende del pasaje ocho, recortar a ciegas a cinco es un falso ahorro.

Otra opción es recuperar de forma amplia y después filtrar con metadatos fiables, búsqueda híbrida o reranking antes de enviar la evidencia al modelo de generación. El reranking tiene su propio coste y solo compensa si recorta suficiente contexto o mejora suficiente la calidad.

No todas las tareas necesitan recuperación. Reformatear la respuesta que acabas de generar probablemente no requiere otra búsqueda vectorial, y el estado de un pedido conocido quizá se obtenga mejor directamente de la API de la aplicación que hace de fuente de verdad que deduciéndolo de documentos.

Para fijar `top_k` a partir de evidencia, registra dos cosas por petición: los pasajes recuperados, con su posición, y los pasajes que la respuesta usó de verdad. Las citas de la respuesta te dan la segunda lista de forma barata; para las respuestas sin citas, revisa una muestra a mano o con un evaluador. A partir de esos registros, sigue la posición del pasaje más profundo que necesitó cada respuesta, agrupada por tipo de consulta. Si las preguntas sobre políticas casi siempre tiran de los tres primeros pasajes mientras que las de resolución de problemas a menudo

llegan a la posición ocho o nueve, da a cada tipo su propio `top_k` en lugar de enviar diez pasajes a todo. Vuelve a comprobar las cifras después de cualquier cambio en el chunking, el modelo de embeddings o el reranker, porque las posiciones se desplazan.

> // CASO · LEDGERLINE.
>
> El agente de investigación enviaba 20 pasajes recuperados con cada pregunta. Registrar qué pasajes citaban de verdad los informes mostró que la mayoría de los tipos de consulta nunca usaba nada más allá del octavo. Con `top_k` fijado por tipo de consulta, y las consultas amplias aún con 20 permitidos, la investigación baja de 640 $ a 560 $ al mes.

// NIVEL 3 En profundidad

El tamaño de los chunks afecta tanto a la calidad de la respuesta como al coste. Los chunks diminutos pueden separar una definición de su excepción, y los grandes pueden arrastrar secciones irrelevantes. Conserva los encabezados, las tablas, los bloques de código y las versiones de los documentos allí donde esos límites importen.

Si la respuesta es incorrecta, revisa lo que se recuperó antes de pasar a un modelo de generación mayor. Un modelo más grande no puede razonar de forma fiable a partir de un párrafo de una política que nunca le llegó. Una mala recuperación también tiende a provocar búsquedas adicionales y reintentos caros.

Un conjunto de referencia pequeño y estable que reutilizan todas las peticiones puede salir más barato como contexto de prompt en caché (Tokens, contexto y caché de prompts) que como un flujo de recuperación aparte. Una colección grande o que se actualiza a menudo suele beneficiarse de una recuperación selectiva.

Costes de los embeddings

Los embeddings se pagan en dos momentos. La ingesta genera el embedding de cada chunk del corpus; el tiempo de consulta genera el de cada pregunta que llega. Los embeddings de consulta son cortos y crecen con el tráfico. Los de ingesta crecen con el tamaño del corpus y con la frecuencia con la que lo reconstruyes.

Los vectores de modelos de embeddings distintos no se pueden comparar, y unos límites de chunk nuevos producen chunks nuevos, así que cambiar el modelo de embeddings o el chunking supone volver a generar los embeddings de todo el corpus. Cada reconstrucción es un coste puntual, pero vuelve cada vez que cambias uno de los dos. Versiona el índice por modelo de embeddings y configuración de chunking, construye la nueva versión junto a la antigua y cambia el tráfico solo cuando pase tus comprobaciones de recuperación.

ILUSTRATIVO: UN CORPUS DE 2 MILLONES DE CHUNKS DE 400 TOKENS CADA UNO SUPONE 2.000.000 × 400 = 800 MILLONES DE TOKENS QUE PROCESAR EN CADA RECONSTRUCCIÓN COMPLETA, ANTES DE CUALQUIER CONSULTA.

El almacenamiento y el tamaño del índice crecen con el número de chunks y las dimensiones de los vectores. Un vector float32 ocupa 4 bytes por dimensión.

ILUSTRATIVO: CON 1.536 DIMENSIONES, UN VECTOR OCUPA 1.536 × 4 = 6.144 BYTES, Y 2 MILLONES DE CHUNKS NECESITAN 6.144 × 2.000.000 ≈ 12,3 GB DE VECTORES EN BRUTO ANTES DE LA SOBRECARGA DEL ÍNDICE. CON 768 DIMENSIONES, EL TAMAÑO EN BRUTO SE REDUCE A LA MITAD, A UNOS 6,1 GB.

Genera los embeddings de ingesta en lotes grandes en lugar de una petición por chunk; si tu proveedor ofrece procesamiento asíncrono con descuento, la ingesta encaja de forma natural (Inferencia por lotes y asíncrona). Guarda un hash del contenido por documento o por chunk y genera embeddings solo de lo que haya cambiado desde la última ejecución, para que una actualización diaria no vuelva a procesar la mayoría que no ha cambiado.

Los modelos de embeddings más pequeños a menudo bastan para la recuperación y reducen a la vez los costes de ingesta, de consulta y de almacenamiento. No lo des por hecho en ningún sentido: ejecuta tu propio conjunto de evaluación (Elige el modelo que necesita la tarea) y compara con qué frecuencia los pasajes que necesita la respuesta caen entre los primeros resultados.

Referencias: Búsqueda web de OpenAI, Búsqueda en ficheros de OpenAI, Precios de OpenAI (llamadas a herramientas y almacenamiento), Embeddings y dimensiones de OpenAI.

Helmcode

// PARTE V

Infraestructura y precios

11 Economía de la inferencia en GPU y del despliegue

VISIÓN GENERAL
PRÁCTICA
EN PROFUNDIDAD Comparar opciones de compra de GPU
El punto de equilibrio de capacidad como un rango

Economía de la inferencia en GPU y del despliegue

Dimensiona la capacidad de GPU para el tráfico pico y los objetivos de latencia antes de compararla con una API de pago por uso.

// NIVEL 1 Visión general

Una instancia de GPU puede parecer barata si divides su precio por hora entre el rendimiento (throughput) teórico en tokens. Ese cálculo supone que el hardware funciona a pleno rendimiento con trabajo útil todas las horas, algo que el tráfico real rara vez permite.

Un modelo puede caber en la memoria de la GPU y aun así tener problemas con peticiones concurrentes de contexto largo. El tráfico puede llegar en picos, y unos requisitos de latencia estrictos pueden obligarte a mantener capacidad sin usar disponible. Autoalojar también te hace responsable de operar el servicio.

Poner un anuncio de alquiler de GPU al lado de una página de precios de una API te dice poco. Compara opciones que den los mismos resultados con el mismo nivel de servicio.

// NIVEL 2 Práctica

Describe la carga de trabajo antes de elegir la infraestructura: longitud de entrada, longitud de salida, concurrencia, objetivos de latencia, picos de tráfico y requisitos de disponibilidad. Sigue el tiempo hasta el primer token, la tasa de tokens generados, el tiempo en cola, la latencia de cola y las tareas aceptadas completadas.

En un modelo denso de 32.000 millones de parámetros, el almacenamiento bruto de los pesos ronda los 64 GB con precisión de 16 bits, 32 GB con 8 bits y 16 GB con 4 bits, antes de los metadatos de cuantización, los buffers de ejecución, la KV cache y otras sobrecargas. En un modelo mixture-of-experts, planifica la memoria para el **total** de los pesos, no solo para los que se activan por token.

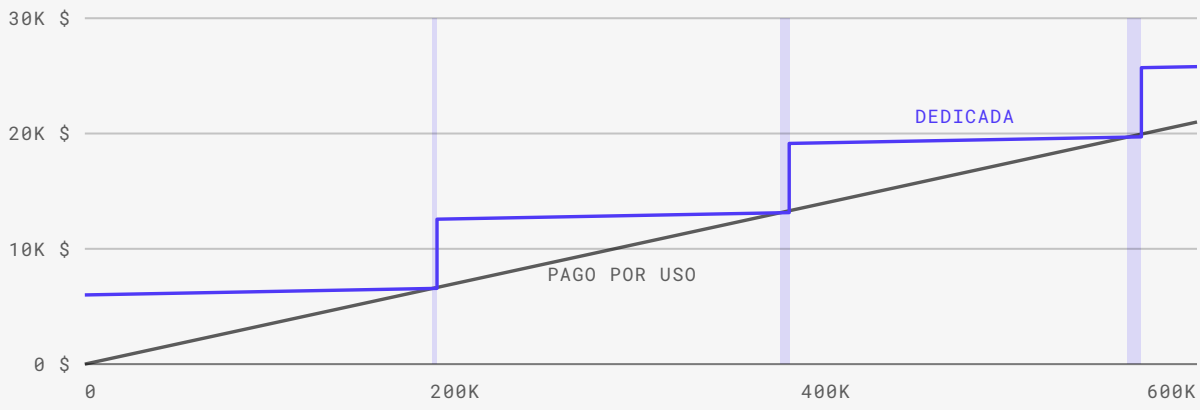
Compara tres opciones:

	API DE PAGO POR USO	CAPACIDAD DEDICADA (GESTIONADA)	AUTOALOJADO
Patrón de precio	Sigue al uso	Recurrente o comprometido	Hardware + operaciones
Riesgo de capacidad ociosa	Normalmente del lado del proveedor	Depende del plan	De tu lado
Responsabilidad del serving	Proveedor	Proveedor	Tu equipo
Control	Limitado	Depende del contrato	Mayor

Supón que un despliegue cuesta 6.000 $ al mes y una API de pago por uso cuesta 0,035 $ por tarea aceptada. Si autoalojar añade 0,003 $ de coste variable por tarea, el volumen de equilibrio simplificado es:

$$N = \frac{6.000}{0{,}035 - 0{,}003} = 187.500 \text{ tareas aceptadas/mes}$$

Esto solo se cumple mientras el despliegue pueda asumir ese volumen sin añadir capacidad y ambas opciones den una calidad y un servicio equivalentes. Los costes reales de GPU suelen subir a escalones a medida que añades réplicas.



Coste mensual frente a volumen con las cifras de la hoja de cálculo del capítulo. La capacidad dedicada solo es más barata en las franjas sombreadas antes de cada nueva réplica; a partir de 190.000 tareas, una segunda réplica duplica el coste fijo.

// CASO · LEDGERLINE.

> Ledgerline calculó el precio de una réplica dedicada para la extracción y lo dejó enseguida. Con 40.000 facturas al mes, toda la línea de extracción cuesta 567 $, mientras que una sola réplica en el ejemplo del capítulo cuesta 6.000 $ antes de cualquier tráfico. A unos 0,014 $ por factura, una réplica necesitaría más de 500.000 facturas al mes para amortizarse, muy por encima de las 190.000 que puede servir, así que se queda la API de pago por uso. La hoja de cálculo facilita repetir las cuentas si cambia la combinación.

// NIVEL 3

En profundidad

Durante el **prefill**, el modelo procesa la secuencia de entrada. Durante el **decode**, genera los tokens de salida. La planificación y la presión sobre la memoria son distintas en las dos fases. El batching continuo puede mejorar el rendimiento útil coordinando peticiones de distinta longitud; la gestión paginada de la KV cache reduce la memoria asignada que se desperdicia; el prefix caching puede reutilizar la KV cache de un prefijo compartido entre peticiones y evitar repetir su prefill (consulta Tokens, contexto y caché de prompts).

La cuantización puede reducir el almacenamiento de los pesos y, a veces, mejorar el rendimiento en hardware compatible, pero puede cambiar el comportamiento del modelo y no hace que la generación salga gratis. Una huella de pesos menor solo ayuda si lo que limita el despliegue es la memoria de los pesos.

Con el despliegue de 6.000 $ al mes del ejemplo de la sección Práctica, 100.000 tareas aceptadas asignan 0,06 $ por tarea. Gestionar 160.000 con el mismo hardware asigna 0,0375 $. La factura no cambia, pero cada tarea cuesta menos. Si se fuerza demasiado, una mayor utilización empeora las colas y la latencia P95, así que un servicio fiable puede necesitar algo de capacidad de reserva.

Incluye en el coste total la conmutación por error, la red, la monitorización, las actualizaciones, la seguridad y el tiempo de ingeniería asignado. Si la migración cuesta 24.000 $ y ahorra 2.000 $ al mes en régimen estable, el periodo de recuperación simple es de 12 meses, antes de otros riesgos y cambios.

Comparar opciones de compra de GPU

La capacidad de GPU en la nube se vende normalmente de tres formas. La capacidad bajo demanda es la más cara por hora y se puede liberar en cualquier momento, aunque puede que el tipo de GPU que quieres no esté disponible cuando lo pidas. La capacidad reservada o comprometida cuesta menos por hora a cambio de un plazo, y la pagas llegue tráfico o no. La capacidad spot o interrumpible es aún más barata, pero el proveedor puede recuperarla con poco aviso.

Ese riesgo de interrupción decide dónde encaja spot. Los trabajos por lotes, la evaluación offline y otro trabajo reintentable pueden absorber un nodo recuperado reanudando desde un checkpoint o volviendo a encolar. El serving crítico en latencia normalmente no puede, salvo que la capacidad bajo demanda o reservada cubra por sí sola el pico.

Pon precio a nodos completos. Una cifra por hora de GPU deja fuera la CPU y la memoria del sistema, el almacenamiento de los pesos del modelo, la red y la transferencia de datos. Después convierte el precio del nodo en coste por tarea aceptada con el rendimiento que hayas medido para tu carga de trabajo. Dos GPU pueden

intercambiar su posición en la clasificación cuando se incluyen el rendimiento y el batching. Los precios y las estructuras de descuento cambian a menudo, así que revisa las páginas de precios vigentes de cada proveedor, como la de precios de AWS EC2 enlazada abajo, antes de hacer la comparación.

El punto de equilibrio de capacidad como un rango

Reúne la forma de la demanda y los supuestos financieros en una sola hoja de cálculo:

VARIABLE	VALOR ILUSTRATIVO	POR QUÉ IMPORTA
Tareas aceptadas al mes	`180.000`	El denominador es el trabajo útil
Coste de la API de pago por uso por tarea aceptada	`0,035 $`	Incluye las llamadas y recuperaciones relevantes
Capacidad fija dedicada	`6.000 $/mes`	Se paga incluso en los periodos ociosos
Coste variable dedicado por tarea	`0,003 $`	Egress, almacenamiento u otros costes por uso
Migración puntual	`24.000 $`	No forma parte del gasto recurrente
Capacidad dedicada disponible	`190.000 tareas/mes`	Supuesta con la latencia pico requerida

Con 180.000 tareas, la alternativa de pago por uso es `180.000 × 0,035 $ = 6.300 $/mes`. La dedicada es `6.000 $ + 180.000 × 0,003 $ = 6.540 $/mes`: **más cara**, incluso antes de la migración. Con 190.000 tareas, el pago por uso sale a 6.650 $ y la dedicada a 6.570 $. Pero con 220.000 tareas el despliegue fijo de este ejemplo se queda sin capacidad, así que no puedes extrapolar el ahorro aparente sin añadir otra réplica o cambiar el requisito de servicio.

En conjunto, la capacidad dedicada de este ejemplo solo gana en una franja estrecha, de 187.500 a 190.000 tareas al mes, antes de que haga falta una segunda réplica. Por debajo de la franja, la API de pago por uso es más barata, y por encima la segunda réplica añade otros 6.000 $ de coste fijo.

El tráfico medio mensual no basta por sí solo para fijar la capacidad. Dimensiona para la tasa de llegada pico, la distribución de longitudes de entrada y de salida, el contexto por petición, la concurrencia y la latencia P95. Una oferta de capacidad dedicada o de un plan de tarifa plana debería detallar el rendimiento y el comportamiento de la cola, además de los cupos mensuales de tokens.

Al comparar una migración, separa el ahorro neto recurrente del coste puntual; El manual de optimización cubre el periodo de recuperación. Trata cualquier punto de equilibrio como el límite de un escenario.

Antes de comprometerte con capacidad dedicada, vuelve a ejecutar la hoja de cálculo con tus propios volúmenes, carga pico y objetivos de latencia. La calculadora de abajo parte de las cifras de la tabla; lleva el volumen más allá de la capacidad de una réplica para ver el escalón.

El mismo cálculo, con una tabla de volúmenes, está en la hoja de cálculo descargable.

Referencias: vLLM, In-flight batching y KV cache paginada en TensorRT-LLM, Artículo de PagedAttention, Precios de AWS EC2, Avisos de interrupción de EC2 Spot.

Helmcode

// PARTE VI

Llevarlo a la práctica

12 La economía de una funcionalidad de IA

VISIÓN GENERAL

PRÁCTICA
- Empieza por la unidad facturable o valiosa
- Una suscripción puede ocultar usuarios caros
- Decide quién asume el riesgo del uso
- Una hoja de cálculo de economía unitaria

EN PROFUNDIDAD
- El margen cambia cuando cambia el producto
- El uso marginal sigue costando capacidad en un plan de tarifa plana
- Evita usar una métrica de coste como KPI de producto
- Pasa la hoja de cálculo por una tabla de sensibilidad

13 El manual de optimización

VISIÓN GENERAL

PRÁCTICA

EN PROFUNDIDAD

La economía de una funcionalidad de IA

Una llamada de inferencia puede ser barata y la funcionalidad construida sobre ella puede seguir perdiendo dinero.

// NIVEL 1 Visión general

Piensa en dos productos. Uno cobra a empresas por cada documento que procesa con éxito. El otro vende una suscripción mensual fija y deja que los clientes usen un asistente de IA tantas veces como quieran.

Usan los mismos modelos a los mismos precios de API negociados, pero sus resultados financieros pueden ser muy distintos.

En el producto de documentos, los ingresos crecen con el trabajo completado, siempre que el precio y el coste directo se mantengan en proporción. En el producto de suscripción, unas pocas cuentas muy activas pueden generar la mayor parte del gasto en inferencia y pagar lo mismo que todos los demás.

Aquí la optimización de costes pasa a ser un problema de economía del producto además de uno de infraestructura. Necesitas saber **qué acción del cliente crea valor, cuánto cuesta servirla y cómo cambian los ingresos a medida que crece el uso**.

Separa el margen bruto de toda la empresa del margen de contribución de una funcionalidad de IA concreta. El segundo es una herramienta de decisión: puede mostrar qué patrones de uso son sostenibles antes de incluir los gastos generales de la empresa y otras decisiones contables.

// NIVEL 2 Práctica

Empieza por la unidad facturable o valiosa

En un procesador de documentos, la unidad puede ser un documento aceptado; en un agente de investigación, un informe completado; en un producto de programación, una tarea aceptada. En un asistente de IA vendido como parte de un producto SaaS más amplio, las unidades relevantes pueden ser el **coste por cuenta activa al mes** y el **coste por interacción útil**.

Usa al menos dos vistas de la economía:

> Contribución por tarea = Ingresos netos atribuibles a la tarea − Coste variable de entregar la tarea

> Contribución por cuenta =
>
> Ingresos netos de la cuenta − Costes de servicio y soporte atribuibles a la cuenta

Ingresos netos significa los ingresos después de los descuentos, créditos y reembolsos aplicables, no un precio de lista de marketing. El coste de servicio debe incluir la inferencia, las herramientas de pago, la infraestructura variable y la corrección humana esperada. Si repartes capacidad fija compartida, etiqueta ese reparto por separado para que el margen variable no se mezcle con el coste unitario totalmente repartido.

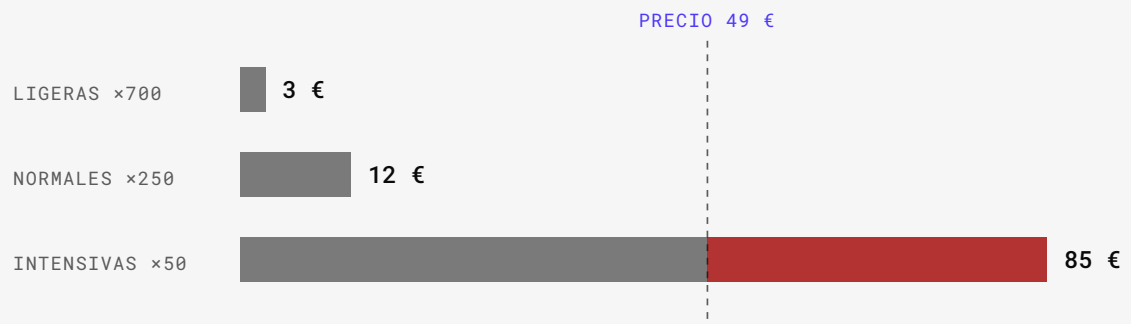
Una suscripción puede ocultar usuarios caros

Un producto de IA hipotético cobra 49 € por cuenta al mes, sin impuestos. Su uso se distribuye así:

GRUPO DE CUENTAS	CUENTAS	COSTE MEDIO DE SERVIR IA POR CUENTA	COSTE TOTAL DE SERVICIO
Usuarios ligeros	700	3 €	2.100 €
Usuarios habituales	250	12 €	3.000 €
Usuarios intensivos	50	85 €	4.250 €
Total	**1.000**		**9.350 €**

Con 1.000 cuentas de pago, los ingresos por suscripción son `1.000 × 49 € = 49.000 €/mes`. Los costes de servir la IA son 9.350 €, alrededor del 19,1 % de los ingresos, lo que en conjunto parece asumible.

Sin embargo, las cincuenta cuentas intensivas generan 4.250 € de coste frente a 2.450 € de ingresos. Su contribución en el servicio de IA es de **−1.800 €** antes de soporte al cliente y otros gastos, y el resto de cuentas las subvencionan.



Coste de servir IA por cuenta y mes frente a una suscripción de 49 €. Las 50 cuentas intensivas cuestan 36 € más de lo que pagan cada una, y el resto de cuentas cubre la diferencia.

Los usuarios intensivos pueden merecer la pena igualmente, porque pueden quedarse más tiempo, recomendar el producto a otros clientes o pasar a un plan superior más adelante. Aun así, trata la subvención como una decisión explícita de precios y de capacidad. Una media sobre todas las cuentas oculta esta cola, igual que lo hace una media sobre las tareas de un agente.

Decide quién asume el riesgo del uso

Los modelos de precios reparten el riesgo de forma distinta:

PRECIO DEL PRODUCTO	QUÉ ESCALA CON EL USO	RIESGO ECONÓMICO
Por tarea completada	Los ingresos suelen seguir al trabajo útil	Hace falta una definición transparente de aceptación y de coste
Por token o crédito	Los ingresos siguen al consumo medido	A los clientes les puede costar predecir el gasto total
Suscripción fija	Los ingresos siguen sobre todo a las cuentas	La cola de usuarios intensivos puede desbordar unos ingresos fijos
Suscripción + uso incluido	Ingresos base más un derecho de uso acotado	Las reglas de excedente y la experiencia de usuario deben estar claras
Precio por puesto	Los ingresos siguen a las licencias	El consumo de inferencia por puesto puede variar muchísimo

Los créditos requieren cuidado. Un «crédito» puede representar una llamada al modelo, un número fijo de tokens o una unidad abstracta de trabajo. Si cambian los costes de los modelos subyacentes, lo que vale económicamente un crédito puede desviarse, salvo que el derecho de uso esté claro y versionado.

A menudo funciona medir el **trabajo valioso** en la capa de producto y seguir los tokens por debajo como impulsor del coste. Los clientes entienden un documento procesado o un informe completado con más facilidad que la contabilidad interna de tokens de razonamiento. Pero el tamaño de las tareas no siempre es estable, así que el producto puede necesitar niveles o reglas de uso razonable para el trabajo inusualmente caro.

Una hoja de cálculo de economía unitaria

Supón que una funcionalidad de informes de investigación tiene estas características hipotéticas:

- Ingresos netos atribuibles a un informe aceptado: 2,50 €.
- Coste medio de inferencia y herramientas por informe enviado: 0,90 €.
- Tasa de aceptación: 80 %.
- Coste adicional esperado de revisión y entrega por informe aceptado: 0,30 €.

Para entregar 100 informes aceptados con una tasa de aceptación del 80 %, el volumen enviado esperado, en el caso simple, es `100 / 0,80 = 125`, suponiendo que la tasa es estable.

El gasto estimado en inferencia y herramientas es `125 × 0,90 € = 112,50 €`. Suma `100 × 0,30 € = 30 €` en costes de revisión y entrega: 142,50 € en total, o **1,425 € por informe aceptado**. Frente a 2,50 € de ingresos, la contribución es de 1,075 € por informe aceptado antes de otros costes repartidos.

Comparar el precio de 2,50 € solo con los 0,90 € por tarea enviada sobrestima el margen, porque los intentos rechazados e incompletos forman parte del coste de entregar el trabajo aceptado.

La hoja de cálculo reproduce este cálculo y la tabla de sensibilidad de En profundidad.

> `// CASO · LEDGERLINE.`
>
> Ledgerline vende informes de investigación dentro de un plan mensual de 299 $. Cada informe aceptado cuesta ahora 0,28 $ en inferencia (560 $ / 2.000). Treinta de sus 400 cuentas generan 1.200 de los 2.000 informes mensuales, 40 cada una, o 11,20 $ de inferencia por cuenta: muy por debajo del precio. Para que siga siendo así si las cuentas intensivas automatizan informes diarios, el plan incluye ahora 200 informes al mes (56 $ de inferencia) y cobra los que pasen de ahí.

`// NIVEL 3` En profundidad

El margen cambia cuando cambia el producto

La distribución de costes de una aplicación de IA rara vez es fija. Añadir acceso a herramientas puede convertir un asistente barato en un flujo de varios pasos. Una ventana de contexto más grande puede cambiar el gasto en prefill. Una mejor calidad del modelo puede reducir los reintentos pero subir el coste por llamada. Un nuevo cliente enterprise puede enviar documentos más grandes o tener una concurrencia pico mucho mayor que los usuarios actuales.

Sigue cada cohorte de cuentas en ingresos netos, trabajo aceptado, costes variables de servir la IA, uso de herramientas de pago, la fracción de usuarios que activan fallbacks y el gasto P95 por cuenta. Los ingresos por cuenta por sí solos no mostrarán que una cohorte se está desplazando hacia la cola de usuarios intensivos.

El uso marginal sigue costando capacidad en un plan de tarifa plana

Con un plan de tarifa plana, una petición elegible adicional puede no subir directamente la factura mensual, pero sigue ocupando rendimiento (throughput). Puede afectar a la latencia de otros usuarios o exigir un nivel superior. Un producto con uso ilimitado de cara al cliente puede, por tanto, consumir capacidad rápidamente mientras la facturación del proveedor sigue siendo predecible.

Mantén una vista de capacidad junto a la financiera. La factura explica el gasto actual, pero no te dirá si el modelo operativo puede soportar demanda adicional.

Evita usar una métrica de coste como KPI de producto

El coste por tarea aceptada se puede optimizar de formas que empeoran el producto. Unas reglas de aceptación tan estrictas que casi nada las cumple hacen que la economía unitaria parezca terrible, y unas reglas que aceptan cualquier respuesta hacen que parezca maravillosa. Ninguna de las dos te dice si los clientes obtuvieron lo que querían.

Vincula la aceptación a resultados reales del producto siempre que puedas. En una herramienta que usan los empleados, ten en cuenta el tiempo humano ahorrado o perdido. En una funcionalidad de cara al cliente, sigue si el resultado fue utilizable, si hizo falta otra ejecución y si los usuarios siguen valorando la funcionalidad. Algunas de estas señales no se convertirán en cifras precisas en dólares; dejarlas fuera hace que los números parezcan más seguros de lo que son.

Pasa la hoja de cálculo por una tabla de sensibilidad

Toma la hoja de cálculo de informes de investigación y varía dos entradas: la tasa de aceptación y el coste de inferencia y herramientas por informe enviado. Los ingresos netos se mantienen en 2,50 € y la revisión en 0,30 € por informe aceptado.

$$\text{Coste por informe aceptado} = \frac{\text{Coste de inferencia y herramientas por informe enviado}}{\text{Tasa de aceptación} + 0{,}30\ €}$$

TASA DE ACEPTACIÓN	CONTRIBUCIÓN CON 0,90 € POR INFORME ENVIADO	CONTRIBUCIÓN CON 1,20 € POR INFORME ENVIADO
60 %	0,700 €	0,200 €
80 %	1,075 €	0,700 €
90 %	1,200 €	0,867 €

ILUSTRATIVO: ARITMÉTICA DE ESCENARIOS BASADA EN LA HOJA DE CÁLCULO ANTERIOR, ANTES DE COSTES REPARTIDOS.

Una subida del coste de 0,90 € a 1,20 € por informe enviado recorta la contribución con un 80 % de aceptación de 1,075 € a 0,700 €. Una caída de la aceptación del 80 % al 60 % hace lo mismo al precio actual. Si ocurren las dos cosas, quedan 0,20 €, el 8 % de los ingresos. Añade filas para las entradas a las que tu producto está más expuesto, como la proporción de tareas que usan una herramienta premium, y una para el nivel de uso que estás planificando.

El manual de optimización

Seis pasos para encontrar los pocos cambios que más van a bajar tu factura.

// NIVEL 1 Visión general

No te recomendamos implementar todas las técnicas de esta guía; podría llevarte meses. Algunos cambios eliminan mucho trabajo innecesario. Otros no ahorran casi nada o hacen que el sistema sea más difícil de mantener.

Empieza por la factura y después mira el flujo de trabajo. Separa la inferencia, la recuperación y las herramientas, la infraestructura, las suscripciones y los gastos operativos, atribuye esos costes a funcionalidades y tareas aceptadas, y solo entonces elige qué cambiar.

// NIVEL 2 Práctica

Trabaja en este orden:

01 **Encuentra las operaciones caras.** Identifica los mayores costes totales y el peor coste por tarea aceptada. Mira los casos atípicos caros además de la media.

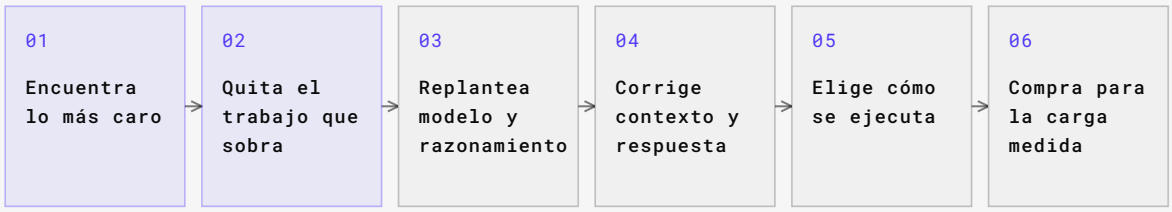
02 **Elimina el trabajo que no debería ocurrir.** Las peticiones duplicadas, las búsquedas repetidas, los cálculos innecesarios con un LLM y la salida que nadie usa a menudo se pueden corregir sin cambiar de proveedor.

03 **Replantea las elecciones de modelo y de razonamiento.** Usa modelos más pequeños adecuados para tareas acotadas; añade enrutamiento o cascadas solo cuando tengan sentido económico.

04 **Corrige el contexto y la forma de la respuesta.** Conserva la evidencia necesaria, quita el material irrelevante, preserva los prefijos reutilizables y compacta el estado del agente de forma deliberada.

05 **Elige cómo se ejecuta el trabajo.** Procesa por lotes el trabajo que puede esperar. Compara una API de pago por uso, un plan de tarifa plana y capacidad dedicada, gestionada o autoalojada, frente a la carga de trabajo que queda.

06 **Compra para la carga de trabajo que has medido.** Cuando el volumen y la forma de las peticiones sean estables, mira los descuentos por compromiso de uso o enterprise, los tramos por volumen, el rendimiento (throughput) aprovisionado o reservado, los niveles de servicio prioritarios frente a los flexibles y las tarifas más altas que algunos proveedores cobran por contexto largo. Comprométete solo con el uso que ya has observado, deja margen para cambios de modelo y de tráfico, y compara el coste total del compromiso con la línea base de pago por uso para el mismo trabajo.



El orden importa: medir y quitar el trabajo innecesario va antes de cualquier cambio de modelo o de proveedor, y suele ser lo que antes se amortiza.

Prioriza por impacto económico neto, confianza, esfuerzo de implementación y riesgo. Como ilustración, eliminar una extracción duplicada podría ahorrar 400 $ al mes con poco trabajo de ingeniería, mientras que una migración a GPU promete 1.500 $ pero conlleva una carga de mantenimiento grande y depende de la utilización. La cifra más llamativa no debería decidir por sí sola qué va primero.

Antes de aprobar cualquier optimización, pregunta:

- ¿Qué trabajo elimina o abarata este cambio?
- ¿Qué parte del flujo de trabajo total representa ese trabajo?
- ¿Qué puede empeorar: corrección, latencia, fiabilidad, privacidad o mantenibilidad?
- ¿Qué nuevo coste recurrente introduce el cambio?
- ¿Estamos comparando resultados aceptados equivalentes?

Apruébala solo si las respuestas muestran que baja el coste de un trabajo aceptado equivalente una vez contados sus nuevos costes.

La hoja de cálculo tiene una hoja para ordenar los cambios candidatos por ahorro, plazo de recuperación, confianza, esfuerzo y riesgo, con estas preguntas como columnas.

Revisa también las suscripciones de IA que están fuera de tu factura de inferencia: asistentes para desarrolladores, herramientas de escritura e investigación, productos para reuniones, licencias de automatización. Cancelar diez licencias sin usar de 30 $/mes en un contrato cancelable ahorra 300 $ al mes, a veces con menos esfuerzo que rediseñar parte de la pila de inferencia.

> // CASO · LEDGERLINE.
>
> Las partidas de inferencia de Ledgerline pasaron de 1.200 $, 600 $ y 800 $ a 567 $, 330 $ y 560 $: 1.457 $ al mes, un 44 % menos. Una herramienta de observabilidad y las ejecuciones periódicas de evaluación añaden 140 $ al mes, así que el total mensual es de 1.597 $ y el ahorro neto de 1.003 $, un 38,6 %. El trabajo costó unos 8.000 $ de tiempo de ingeniería, que se recuperan en ocho meses. Medir fue lo primero: sin el informe por operación del capítulo 2, el equipo habría empezado por el modelo del agente de investigación en lugar de por sus ejecuciones descontroladas.

// NIVEL 3 En profundidad

Mide el **ahorro neto** sobre el total, para que un porcentaje nunca caiga sobre el denominador equivocado:

$$\text{Ahorro neto mensual} = \text{Total mensual de la línea base} - \text{Total mensual optimizado}$$

$$\text{Plazo de recuperación (meses)} = \frac{\text{Coste único}}{\text{Ahorro neto mensual}}$$

El plazo de recuperación solo tiene sentido cuando el ahorro neto mensual es positivo. Deja los costes únicos (migración, tiempo de ingeniería, penalizaciones por salida de contrato) fuera de la cifra mensual y muéstralos a través del plazo de recuperación. Si la inferencia baja de 4.000 $ a 2.800 $ pero el cambio añade 300 $ de herramientas y 400 $ de operaciones, el gasto mensual pasa a ser de 3.500 $. La partida de inferencia bajó un 30 %; el total bajó un **12,5 %**, un ahorro neto de 500 $ al mes. Si el cambio supuso 4.000 $ de trabajo de ingeniería, el plazo de recuperación es de 8 meses.

Un panel de presupuesto puede avisarte, pero los límites de gasto de los agentes se tienen que aplicar en el código de la aplicación, como se describe en Bucles de agentes, validación y reintentos.

Mantén una rutina ligera. Revisa a menudo las ejecuciones inusuales, mira el coste por funcionalidad y por proveedor cada mes, y replantea las suscripciones, los compromisos y la infraestructura con menos frecuencia. Los cambios grandes de tráfico, de modelos o de comportamiento del producto son motivo para revisar tus supuestos.

Referencias: Niveles Flex y Priority de Gemini, Procesamiento Flex de OpenAI, Precios de Gemini por longitud de contexto.

Helmcode

// PARTE VII

Nota del proveedor

14 Cuándo tiene sentido la inferencia de tarifa plana
VISIÓN GENERAL
PRÁCTICA
EN PROFUNDIDAD

Cuándo tiene sentido la inferencia de tarifa plana

Qué incluyen los planes de Helmcode, dónde terminan y cómo saber si uno de ellos bajaría tu factura.

CAPÍTULO DEL PROVEEDOR

// NIVEL 1 Visión general

Una vez que has eliminado las llamadas innecesarias y elegido modelos adecuados, la pregunta que queda es cómo pagar el trabajo que hace tu aplicación.

Las API de pago por uso se adaptan a cargas de trabajo variables. La capacidad dedicada puede ser económica con una demanda predecible, pero alguien tiene que operarla. Un plan de tarifa plana cobra una cuota mensual fija por un servicio definido, con asignaciones por modelo y límites de operación.

Ningún servicio de inferencia tiene capacidad ilimitada, diga lo que diga un plan. La pregunta útil es si tu carga de trabajo encaja en los modelos, las asignaciones y los rate limits (límites de tasa) de un plan, y si la cuota fija es inferior al gasto de pago por uso que sustituye.

// NIVEL 2 Práctica

Helmcode ejecuta modelos de pesos abiertos en infraestructura que opera en Europa, detrás de una API compatible con OpenAI. Los modelos Qwen y Gemma no tienen tope mensual de tokens. DeepSeek V4.1 Flash y GLM 5.3 Flash comparten una asignación mensual que depende del plan.

Planes a noviembre de 2026:

PLAN	PRECIO MENSUAL	API KEYS	ASIGNACIÓN MENSUAL COMPARTIDA DE DEEPSEEK V4.1 FLASH / GLM 5.3 FLASH
Starter	399 €	5	5.000 millones de tokens
Growth	1.299 €	15	15.000 millones de tokens
Scale	3.199 €	40	35.000 millones de tokens
Enterprise	A medida	A medida	Condiciones a medida

Cada API key, incluidas las keys del complemento GLM 5.3, está limitada a **2 millones de tokens por minuto (2M TPM)**, y también se aplican límites de peticiones por minuto y de concurrencia. GLM 5.3 es un complemento de 150 € por API key habilitada al mes, con su propia asignación mensual, separada de la compartida. **Una asignación mensual no se puede ampliar una vez agotada**, y no hay cargos por excedente a los que recurrir. Los modelos propietarios disponibles a través de la plataforma no forman parte del plan de tarifa plana.

Toma una empresa hipotética que gasta 1.000 € al mes en inferencia de pago por uso. Si 800 € de ese gasto pueden pasar a modelos cubiertos por Starter y 200 € se quedan con el proveedor original, el total modelado es 399 € + 200 € = 599 €: una diferencia bruta de **401 € al mes** antes de los costes de migración, de fallback y otros. Esto supone que la carga de trabajo cumple sus requisitos de calidad y de capacidad con los nuevos modelos.

Es poco probable que una aplicación que gasta solo 120 € al mes vea una factura más baja con un plan de 399 €, salvo que el plan sustituya otros costes o genere valor en otra parte.

> // CASO · LEDGERLINE.
>
> El gasto de pago por uso que le queda a Ledgerline en extracción y soporte es de 897 $ al mes; el agente de investigación sigue con su modelo propietario actual. Antes de plantearse un plan de tarifa plana, el equipo comprobaría que los modelos abiertos que cubre superan los conjuntos de evaluación de extracción y de soporte, que el tráfico pico se mantiene por debajo de 2 millones de tokens por minuto por key, y que un plan Starter de 399 €, convertido al tipo de cambio del día, cuesta menos que la parte de esos 897 $ que de verdad se trasladaría.

// NIVEL 3 En profundidad

La lógica del punto de equilibrio de Economía de la inferencia en GPU y del despliegue se aplica, con la cuota del plan como coste fijo:

$$\text{Volumen de equilibrio} = \frac{\text{Cuota fija mensual}}{\text{Coste variable actual por tarea} - \text{Coste variable restante por tarea}}$$

Con 399 € fijos, un coste actual de 0,020 € por tarea aceptada y un coste variable restante de 0,002 €, el punto de equilibrio simplificado es de unas **22.167 tareas al mes**. El cálculo deja de valer si la carga de trabajo necesita un nivel superior, cargos adicionales por modelo o capacidad adicional.

Una asignación mensual de tokens dice poco sobre la tasa pico de peticiones. Comprueba por separado las peticiones por minuto, los tokens por minuto, la concurrencia por key y los límites de contexto, frente a tu hora de más actividad y no frente a tu media mensual.

El desarrollador del modelo y el operador de la inferencia son papeles distintos. Helmcode ejecuta sus modelos abiertos en la UE sin registro de prompts; los modelos propietarios a los que se accede a través de la misma API los operan sus proveedores y pueden tener condiciones de tratamiento distintas. Comprueba la ruta de cada modelo en lugar de suponer que una sola API implica un solo conjunto de condiciones de residencia.

Un plan de tarifa plana encaja con cargas de trabajo sostenidas que se mantienen dentro de sus límites. No elimina la necesidad de recortar el trabajo desperdiciado: las llamadas innecesarias siguen consumiendo asignación y rendimiento.

Fuentes: Precios de Helmcode, modelos, rate limits. Precios y límites a noviembre de 2026.

// REFERENCIA

Glosario

Amplificación de reintentos TAMBIÉN RETRY AMPLIFICATION

Reintentos que se acumulan entre capas: un SDK que lo intenta tres veces dentro de un bucle de la aplicación que lo intenta tres veces puede enviar nueve peticiones aguas arriba. Fija un único plazo global y conoce la política de cada capa.

VER CAP. 09

API de pago por uso TAMBIÉN METERED API

Inferencia facturada por uso, normalmente por token de entrada y de salida. Es sencilla de operar y la factura sigue al trabajo, lo que encaja con el tráfico variable.

VER CAP. 11 · CAP. 05

Batch API TAMBIÉN API POR LOTES

Un modo del proveedor para peticiones que pueden esperar, a menudo horas, normalmente a un precio inferior al de las llamadas en tiempo real. Es un nivel de precios. Ejecutar trabajo de forma asíncrona en tu propio sistema y agrupar por lotes en una GPU son cosas distintas.

VER CAP. 08

Batching continuo TAMBIÉN PROCESAMIENTO CONTINUO POR LOTES, CONTINUOUS BATCHING

Una técnica del motor de inferencia que intercala peticiones de distinta longitud en la misma GPU para mantenerla ocupada. Aumenta el rendimiento en el hardware que operas y no tiene nada que ver con el descuento por lotes de un proveedor.

VER CAP. 11 · CAP. 08

Caché de prefijos TAMBIÉN PREFIX CACHING

Un motor de inferencia que conserva la KV cache de un prefijo de prompt compartido y la reutiliza en peticiones posteriores que empiezan igual. Ahorra el prefill de ese prefijo; cada token de salida se sigue generando. En hardware dedicado la ganancia aparece como rendimiento, no como una factura más pequeña.

VER CAP. 06 · CAP. 11

Caché de prompts TAMBIÉN PROMPT CACHING

Una funcionalidad del proveedor que cobra o procesa de otra forma un prefijo de entrada repetido. Las escrituras, las lecturas, los TTL y los tamaños mínimos varían según el proveedor, así que revisa los campos de tokens en caché de la respuesta antes de contar ahorros.

VER CAP. 06

Caché de respuestas TAMBIÉN RESPONSE CACHING

Reutilizar un resultado anterior válido para las mismas entradas, con una clave basada en el contenido y las versiones y aislada por tenant. Puede ahorrar una llamada entera. La caché semántica, que empareja entradas parecidas en lugar de idénticas, es más arriesgada.

VER CAP. 07 · CAP. 06

Capacidad dedicada TAMBIÉN CAPACIDAD FIJA, DEDICATED CAPACITY, FIXED CAPACITY

Capacidad de inferencia reservada para ti y pagada por meses, tanto si la opera un proveedor como si la operas tú en tus propias GPU. La factura no sigue al uso, así que el tiempo ocioso lo pagas tú y el crecimiento llega por escalones.

VER CAP. 11

Cascada TAMBIÉN CASCADE

Empezar con un modelo más barato y escalar a uno más potente cuando una comprobación detecta un fallo o un resultado flojo. Compensa si los fallos se detectan de forma fiable y las comprobaciones son baratas.

VER CAP. 04 · CAP. 03

Compactación de contexto TAMBIÉN CONTEXT COMPACTION

Sustituir una transcripción larga por un estado de trabajo más corto: objetivo, restricciones, hallazgos, enfoques fallidos y dónde está la evidencia. Consume tokens en sí misma y puede romper la reutilización del prefijo del prompt, así que hazla cuando la reducción lo justifique.

VER CAP. 07 · CAP. 09

Conjunto de evaluación TAMBIÉN EVAL SET, EVALUATION SET

Un conjunto fijo de casos reales con resultados correctos conocidos, que sirve para comparar modelos o prompts antes de cambiar y para detectar regresiones después de un cambio. Ejecutarlo cuesta dinero, y ese coste forma parte de la decisión.

VER CAP. 03

Coste asignado TAMBIÉN ALLOCATED COST

La parte de una factura fija que se atribuye a cada tarea, como un servicio de 1.000 $ al mes dividido entre las tareas que ha gestionado. Cambia con el volumen aunque la factura no cambie. Compáralo con el coste marginal.

VER CAP. 01 · CAP. 11

Coste marginal TAMBIÉN MARGINAL COST

Lo que una tarea más añade a la factura. En capacidad fija con margen de sobra puede estar cerca de cero; cuando el servicio se satura, la siguiente tarea puede exigir un nivel nuevo.

VER CAP. 01 · CAP. 12

Coste por tarea aceptada TAMBIÉN COST PER ACCEPTED TASK

El coste atribuible total del flujo dividido entre las tareas aceptadas: inferencia, recuperación, herramientas, reintentos, cómputo, corrección humana y una parte de la capacidad fija. Es la métrica central de la guía: el trabajo fallido y abandonado se carga al trabajo que salió bien.

VER CAP. 01 · CAP. 02

Coste total de propiedad TAMBIÉN TCO, TOTAL COST OF OWNERSHIP

Todo lo que cuesta operar un servicio: inferencia o hardware, más failover, red, monitorización, actualizaciones, seguridad y tiempo de ingeniería, con la migración única mostrada aparte.

VER CAP. 11 · CAP. 13

Cuantización TAMBIÉN QUANTIZATION

Guardar los pesos con menos precisión, por ejemplo 8 o 4 bits en lugar de 16. Reduce la memoria y puede aumentar el rendimiento en hardware compatible, pero puede cambiar el comportamiento del modelo.

VER CAP. 11 · CAP. 05

Decode TAMBIÉN DECODIFICACIÓN

La fase en la que un modelo genera los tokens de salida de uno en uno. Exige al hardware de forma distinta que el prefill, y por eso las entradas largas y las salidas largas cuestan distinto de servir.

VER CAP. 11

Destilación TAMBIÉN FINE-TUNING, DISTILLATION

Entrenar un modelo más pequeño para hacer una tarea acotada y estable que hoy resuelve uno más grande. Puede bajar el coste por llamada, y añade costes de entrenamiento, evaluación, alojamiento y reentrenamiento. La licencia tiene que permitirlo.

VER CAP. 03 · CAP. 05

Enrutamiento TAMBIÉN ROUTING

Elegir un modelo para cada petición. Cuando la aplicación ya conoce la operación, una tabla de consulta estática es más sencilla que pedirle a un modelo que enrute a otro modelo.

VER CAP. 04

Fragmentación TAMBIÉN CHUNKING

Cómo se dividen los documentos antes de indexarlos para la recuperación. Los fragmentos demasiado pequeños separan una regla de su excepción; los demasiado grandes meten texto irrelevante en cada petición.

VER CAP. 10

Idempotencia TAMBIÉN IDEMPOTENCY

La propiedad de que repetir la misma operación prevista tenga el mismo efecto que hacerla una vez. Necesita una identidad de operación duradera y, para efectos secundarios como pagos o correos, un contrato que el servicio de destino admita. Es distinta de la caché de respuestas.

VER CAP. 07

KV cache TAMBIÉN CACHÉ KV

El estado de atención que una petición construye al leer su entrada y que luego lee cada paso de decode. Crece con la longitud del contexto y con el número de peticiones concurrentes, y por eso los contextos largos necesitan más memoria de GPU.

VER CAP. 11 · CAP. 06

Margen de contribución TAMBIÉN CONTRIBUTION MARGIN

Los ingresos netos de una tarea o una cuenta menos el coste variable de servirla. Usado por funcionalidad, muestra qué patrones de uso se pagan solos antes de incluir los gastos generales de la empresa.

VER CAP. 12

Mezcla de expertos TAMBIÉN MOE, MIXTURE OF EXPERTS

Una arquitectura de modelo que activa solo una parte de sus parámetros para cada token. Servirlo sigue exigiendo memoria para todos los pesos, así que el número de parámetros activos se queda corto respecto a lo que hace falta para alojarlo.

VER CAP. 05 · CAP. 11

Modelo de pesos abiertos TAMBIÉN OPEN-WEIGHT MODEL

Un modelo cuyos pesos se publican para que otros lo ejecuten, con una licencia que varía según la versión. Descargarlo no cuesta nada, pero ejecutarlo sí lo pagas, a través de una API, de capacidad gestionada o de tus propias GPU.

VER CAP. 05

P95 TAMBIÉN PERCENTIL 95

El valor por debajo del cual queda el 95 % de las observaciones. Se usa para la latencia y para el coste por tarea, porque unas pocas ejecuciones largas pueden llevar la cola muy por encima de la mediana, y la media las oculta.

VER CAP. 01 · CAP. 11

Periodo de recuperación TAMBIÉN PAYBACK, PAYBACK PERIOD

El coste único dividido entre el ahorro neto mensual. Solo tiene sentido cuando el ahorro mensual es positivo; los costes únicos se quedan fuera de la cifra mensual.

VER CAP. 13 · CAP. 11

Plan de tarifa plana TAMBIÉN INFERENCIA DE TARIFA PLANA, FLAT-RATE PLAN, FLAT-RATE INFERENCE

Una cuota periódica por un servicio de inferencia definido, con asignaciones por modelo y límites operativos. Ningún plan de este tipo es ilimitado, así que comprueba que tu carga de trabajo cabe en sus asignaciones, rate limits, concurrencia y latencia.

VER CAP. 14 · CAP. 12

Prefill TAMBIÉN FASE DE PREFILL

La fase en la que un modelo lee la entrada antes de generar. Los prompts largos y el contexto recuperado la encarecen; la caché de prefijos puede evitar repetirla.

VER CAP. 11 · CAP. 06

Presupuesto de razonamiento TAMBIÉN ESFUERZO DE RAZONAMIENTO, REASONING BUDGET, REASONING EFFORT

Un control que exponen algunos modelos de razonamiento para limitar la generación interna antes de la respuesta. Ajústalo por operación; un límite de tokens de salida no lo sustituye y, en su lugar, puede truncar la respuesta.

VER CAP. 04 · CAP. 03

RAG TAMBIÉN GENERACIÓN AUMENTADA POR RECUPERACIÓN, RETRIEVAL-AUGMENTED GENERATION

Recuperar los pasajes relevantes y enviarlos al modelo junto con la pregunta. Su coste abarca la ingesta, los embeddings, el almacenamiento, la recuperación, el reranking y cada token de contexto que añade.

VER CAP. 10

Reranking TAMBIÉN REORDENACIÓN

Una segunda pasada que reordena los pasajes recuperados para que solo lleguen al modelo los más relevantes. La pasada extra tiene un coste, que solo se recupera si recorta suficiente contexto o mejora lo bastante las respuestas.

VER CAP. 10

Showback TAMBIÉN CHARGEBACK

Informar del gasto en IA al equipo o la funcionalidad que lo generó. El chargeback va un paso más allá y lo carga a su presupuesto. Los dos exigen antes etiquetar los costes por equipo o funcionalidad.

VER CAP. 02

Tarea aceptada TAMBIÉN ACCEPTED TASK

Una tarea cuyo resultado cumple los requisitos que definiste para esa operación. En una extracción, por ejemplo: la forma correcta, coherente con las reglas de negocio y correcta respecto a la fuente. Ese listón lo fija tu producto. Una respuesta correcta de la API no basta por sí sola para cumplirlo.

VER CAP. 01 · CAP. 02

Tasa de aceptación TAMBIÉN ACCEPTANCE RATE

Las tareas aceptadas divididas entre las tareas de la cohorte. Muéstrala junto al coste por tarea aceptada, porque cada número por separado oculta el comportamiento. Relajar o endurecer la aceptación puede mover los dos.

VER CAP. 02 · CAP. 12

Tasa de escalado TAMBIÉN ESCALATION RATE

La proporción de peticiones que una cascada pasa al modelo más potente. Junto con el coste de cada comprobación, decide si la cascada sale más barata que usar directamente el modelo más potente.

VER CAP. 04 · CAP. 03

Tokens por minuto TAMBIÉN TPM, TOKENS PER MINUTE

Un rate limit sobre cuántos tokens puede procesar una key o una cuenta cada minuto. Una asignación mensual dice poco de él, así que comprueba por separado los TPM, la tasa de peticiones y la concurrencia.

VER CAP. 14

Volumen de equilibrio TAMBIÉN PUNTO DE EQUILIBRIO, BREAK-EVEN VOLUME

El volumen mensual en el que una opción de coste fijo cuesta lo mismo que una de pago por uso: la cuota fija dividida entre el ahorro variable por tarea. Solo es válido mientras la capacidad, la calidad y el servicio sigan siendo comparables, así que trátalo como el límite de un escenario.

VER CAP. 11 · CAP. 14

Helmcode

// LÉELA EN LÍNEA

Léela en línea, con las tres profundidades de lectura, la calculadora de capacidad y cada actualización.

`helmcode.com/es/guias/optimizacion-costes-ia`

La edición en línea es la de referencia. Este PDF es la edición del 4 nov 2026.

Hoja de cálculo: `helmcode.com/downloads/ai-cost-optimization-worksheet.es.xlsx`

SQL de la línea base: `helmcode.com/downloads/ai-cost-optimization-baseline.sql`

Edición del 4 nov 2026: Primera edición en castellano: introducción y 14 capítulos en tres profundidades de lectura, el caso Ledgerline, nueve figuras, una calculadora de capacidad, un glosario de 38 términos, una hoja de cálculo, una línea base en SQL y el PDF. Precios y límites a noviembre de 2026.