Talutil

Optimizar tokens en Claude: la guía práctica para gastar menos en Opus, Sonnet y Haiku

30 May 2026 · 9 min de lectura

Cada palabra que escribes en Claude tiene un precio, y casi nadie lo está calculando bien. Si usas la API para tu producto, ese precio aparece directamente en tu factura. Si usas la app, se traduce en cuántas conversaciones útiles puedes tener antes de toparte con un límite. En ambos casos el recurso es el mismo: los tokens. Optimizarlos no es tacañería, es la diferencia entre un flujo de trabajo con IA que escala y uno que se vuelve caro o lento sin que entiendas por qué.

Esta guía recoge tácticas que funcionan en todas las versiones de Claude (Haiku, Sonnet y Opus, sin importar el número de generación) para que saques más valor de cada token, pagues menos y mantengas la calidad de las respuestas intacta.

Qué es un token y por qué deberías optimizarlo

Un token es la unidad mínima con la que un modelo de lenguaje procesa texto. No es exactamente una palabra: es un fragmento. En español, una palabra común suele ocupar entre uno y tres tokens, y cosas como tildes, signos o palabras largas elevan la cuenta. Una regla mental útil: alrededor de 750 palabras equivalen a unos 1000 tokens.

Claude cobra (o consume) tokens en dos direcciones:

- **Tokens de entrada (input):** todo lo que envías, incluyendo tu instrucción, el contexto previo de la conversación, los documentos que adjuntas y las instrucciones de sistema.
- **Tokens de salida (output):** todo lo que Claude responde.

El detalle clave que cambia tu estrategia: la salida cuesta mucho más que la entrada. En la línea actual de modelos, el output cuesta aproximadamente cinco veces más que el input. Por eso, controlar la longitud de las respuestas suele ahorrar más que recortar tus preguntas.

Cómo se consumen los tokens en cada versión de Claude

Los precios se expresan por millón de tokens (MTok) y se facturan por separado para entrada y salida. La estructura de la familia Claude se organiza en tres niveles, y entender esta jerarquía es el primer paso de cualquier optimización.

Modelo	Entrada (por millón)	Salida (por millón)	Mejor para
Haiku (4.5)	1.00 USD	5.00 USD	Volumen alto, tareas simples, velocidad
Sonnet (4.6)	3.00 USD	15.00 USD	Equilibrio entre costo e inteligencia
Opus (línea 4.x)	5.00 USD	25.00 USD	Razonamiento complejo, máxima calidad

Estos valores corresponden a la generación 4.x y se han mantenido estables, pero conviene confirmarlos siempre en la página oficial de precios de Anthropic antes de proyectar costos, porque el portafolio se refresca con cierta frecuencia. Un apunte importante: cada versión nueva de Opus puede traer un tokenizador distinto que genera más tokens para el mismo texto, así que el costo efectivo por solicitud puede subir aunque el precio por token no cambie. Conviene medir tu carga real al migrar de una versión a otra.

Elige la versión correcta para cada tarea

La optimización más rentable casi nunca es escribir prompts más cortos. Es usar el modelo correcto. Mucha gente usa el modelo más potente para todo, y eso es como contratar a un arquitecto para colgar un cuadro.

Cuándo usar Haiku

Haiku es el nivel más económico y rápido. Es ideal para clasificar mensajes, extraer datos, generar respuestas cortas, moderar contenido, etiquetar productos o cualquier tarea repetitiva de alto volumen donde no necesitas razonamiento profundo. Si procesas miles de operaciones al día, mover esas tareas a Haiku puede reducir tu factura a una fracción.

Cuándo usar Sonnet

Sonnet es el punto dulce para la mayoría de proyectos. Combina buena calidad de razonamiento con un costo intermedio, y maneja con soltura redacción, análisis, soporte conversacional y flujos con herramientas. Para un negocio que recién automatiza tareas con IA, Sonnet suele ser el modelo por defecto.

Cuándo usar Opus

Opus es el nivel de mayor capacidad. Resérvalo para lo que de verdad lo justifica: razonamiento complejo en varios pasos, decisiones de negocio delicadas, análisis legal o financiero, código difícil o tareas donde un error cuesta más que la diferencia de precio. Usar Opus para resumir un correo es desperdiciar dinero.

Una estrategia avanzada es el enrutamiento por dificultad: usar Haiku o Sonnet para filtrar y resolver lo fácil, y escalar a Opus solo cuando la tarea lo amerita.

Tácticas para reducir los tokens de entrada

El contexto que envías se acumula, sobre todo en conversaciones largas y en aplicaciones que arrastran historial. Estas prácticas lo controlan:

1. **Sé específico, no extenso.** Una instrucción clara de tres líneas rinde más que un párrafo lleno de relleno cortés. Claude no necesita que le pidas las cosas con rodeos.
2. **No repitas contexto innecesario.** Si ya diste una instrucción al inicio, no la reenvíes en cada mensaje.
3. **Resume el historial en conversaciones largas.** Cuando un chat se alarga, pídele a Claude un resumen de los puntos clave y continúa desde ahí, en lugar de arrastrar toda la transcripción.
4. **Adjunta solo lo relevante.** Si un documento tiene cien páginas y solo te interesan dos, pega esas dos. Cada página adjunta es input que pagas.
5. **Reutiliza una buena instrucción de sistema, pero mantenla compacta.** Las instrucciones de sistema viajan en cada solicitud, así que cada palabra extra se multiplica por el número de llamadas.

Tácticas para controlar los tokens de salida

Como la salida es la parte cara, aquí está el mayor margen de ahorro:

- **Pide longitud explícita.** Indica algo como responde en máximo 150 palabras o dame solo la lista, sin introducción. Claude tiende a ser exhaustivo si no le pones un límite.
- **Pide solo el formato que necesitas.** Si quieres una tabla, pide la tabla. Evitas párrafos explicativos que no vas a usar.

- **Evita el reprocesamiento.** Una respuesta bien encuadrada a la primera ahorra dos o tres correcciones, y cada corrección consume input más output otra vez.
- **Desactiva el razonamiento extendido cuando no aporta.** Para tareas simples, el pensamiento paso a paso añade tokens de salida sin mejorar el resultado.

Prompt caching: el ahorro que casi nadie aprovecha

Esta es, probablemente, la palanca más subestimada para quienes usan la API. El prompt caching te permite reutilizar partes fijas de tu contexto (instrucciones de sistema largas, documentos de referencia, ejemplos) sin pagarlas completas en cada llamada.

El funcionamiento en términos de costo es muy favorable: las lecturas de caché cuestan alrededor del diez por ciento de la tarifa base de entrada, lo que representa un ahorro de hasta el noventa por ciento en esos tokens repetidos. La primera escritura en caché cuesta un poco más que el input normal (alrededor de 1.25 veces para la caché de cinco minutos), pero se amortiza rápido si reutilizas ese contexto varias veces.

Un consejo práctico para optimizar tokens en Claude: coloca todo lo estable (instrucciones, referencias) al inicio del prompt y deja lo dinámico al final. Si mezclas un dato variable dentro del bloque que querías cachear, la caché puede fallar de forma silenciosa y terminas pagando todo a precio completo sin darte cuenta. Por eso conviene revisar los registros de uso para confirmar que la caché está funcionando.

Batch API: 50% de descuento para tareas que no son urgentes

Si tienes trabajo que no necesitas resolver al instante (generar descripciones de cientos de productos, clasificar un archivo histórico, traducir un catálogo), el procesamiento por lotes de la API aplica un descuento del cincuenta por ciento tanto en entrada como en salida. Las tareas se procesan de forma asíncrona, normalmente dentro de un plazo de hasta 24 horas.

Lo mejor es que el caching y el batch se acumulan con otras palancas, así que combinar el modelo correcto, caching y batch puede llevar tu costo efectivo muy por debajo de la tarifa de lista.

Optimización en la app de Claude frente a la API

No todos usan la API. Si trabajas en la app de Claude, la lógica de tokens sigue importando, solo que se traduce en límites de uso en lugar de una factura directa. Para aprovecharla mejor:

- Abre un chat nuevo para temas nuevos. Un chat largo arrastra todo el historial como contexto y se agota antes.
- Usa proyectos o instrucciones personalizadas para no repetir el mismo contexto en cada conversación.
- Sé directo con tus peticiones para que cada intercambio cuente.

Si estás en un plan de pago, estas prácticas estiran tu cuota disponible y reducen la sensación de toparte con límites.

Errores comunes que disparan tu consumo de tokens

- Usar Opus para todo por costumbre.
- Pegar documentos enteros cuando solo necesitas un fragmento.
- Mantener conversaciones eternas en lugar de resumir y reiniciar.
- No poner límites de longitud y recibir respuestas el doble de largas de lo necesario.
- Ignorar el caching y el batch cuando el patrón de uso es claramente repetitivo.

- No medir. Sin revisar el consumo real, optimizas a ciegas.

Una rutina de optimización en cinco pasos

Para aterrizar todo lo anterior, aquí tienes un checklist que puedes aplicar a cualquier flujo de trabajo con Claude:

1. **Clasifica la tarea por dificultad** y asígnale el modelo más económico que la resuelva bien.
2. **Recorta el input:** envía solo el contexto necesario y reutiliza instrucciones compactas.
3. **Limita el output:** define longitud y formato desde el prompt.
4. **Activa caching y batch** si tu uso es repetitivo o no urgente.
5. **Mide y ajusta:** revisa el consumo, identifica dónde se va el gasto y corrige.

Optimizar tokens en Claude inteligentemente

Optimizar tokens en Claude no consiste en escribir menos ni en sacrificar calidad. Consiste en tomar decisiones conscientes: el modelo adecuado para cada tarea, contexto limpio, salidas acotadas y el uso inteligente de caching y batch cuando aplica. La buena noticia es que estos principios se mantienen estables en todas las versiones, así que una vez que los interiorizas, seguirán sirviéndote sin importar qué generación de Haiku, Sonnet u Opus uses mañana. Empieza por la palanca más simple, elegir bien el modelo, y construye desde ahí.