# Optimización de Tokens en IA de Lenguaje

Guía práctica para interactuar con modelos de IA de forma precisa, eficiente y sin desperdiciar contexto.

INGENIERÍA DE PROMPTS | MCP · GRAPHIFY | VIBECODE

# Lo que debes saber antes de interactuar con IA

## Cada token cuesta

Los modelos cobran por token de entrada y salida. Un prompt mal estructurado puede multiplicar el coste por 10.

## El contexto es finito

La ventana de contexto tiene un límite. Incluir archivos enteros agota el espacio útil rápidamente.

## La ambigüedad genera ruido

Un prompt vago obliga al modelo a inferir, generando respuestas largas, imprecisas o alucinaciones.

PRINCIPIO FUNDAMENTAL

# Ni más, ni menos: sé preciso

La IA no necesita todo tu proyecto para resolver un problema concreto. Indicarle exactamente qué necesita reduce el consumo de tokens y mejora la calidad de la respuesta.

Un prompt de 200 tokens bien construido puede ser más efectivo que uno de 20.000 tokens con contexto innecesario.


# Instrucciones hiperespecíficas: evita la ambigüedad

### ❌ Vago

«Mejora este código» — El modelo no sabe qué mejorar, en qué archivo ni con qué criterio. Genera respuestas genéricas y consume contexto innecesario.

### ✅ Hiperespecífico

«Refactoriza la función `calculateTotal` en `src/utils/pricing.ts` para manejar descuentos acumulativos. Devuelve solo el código modificado.»

# Herramientas que mapean el código

En lugar de cargar archivos completos, estas herramientas permiten a la IA consultar solo lo que necesita.

## Graphify

Construye un grafo de conocimiento del proyecto. La IA navega por nodos y relaciones en lugar de leer archivos enteros.

## MCP

Protocolo que permite a la IA ejecutar herramientas externas y consultar recursos sin inyectar todo el contexto en el prompt.

## MCP + Graphify

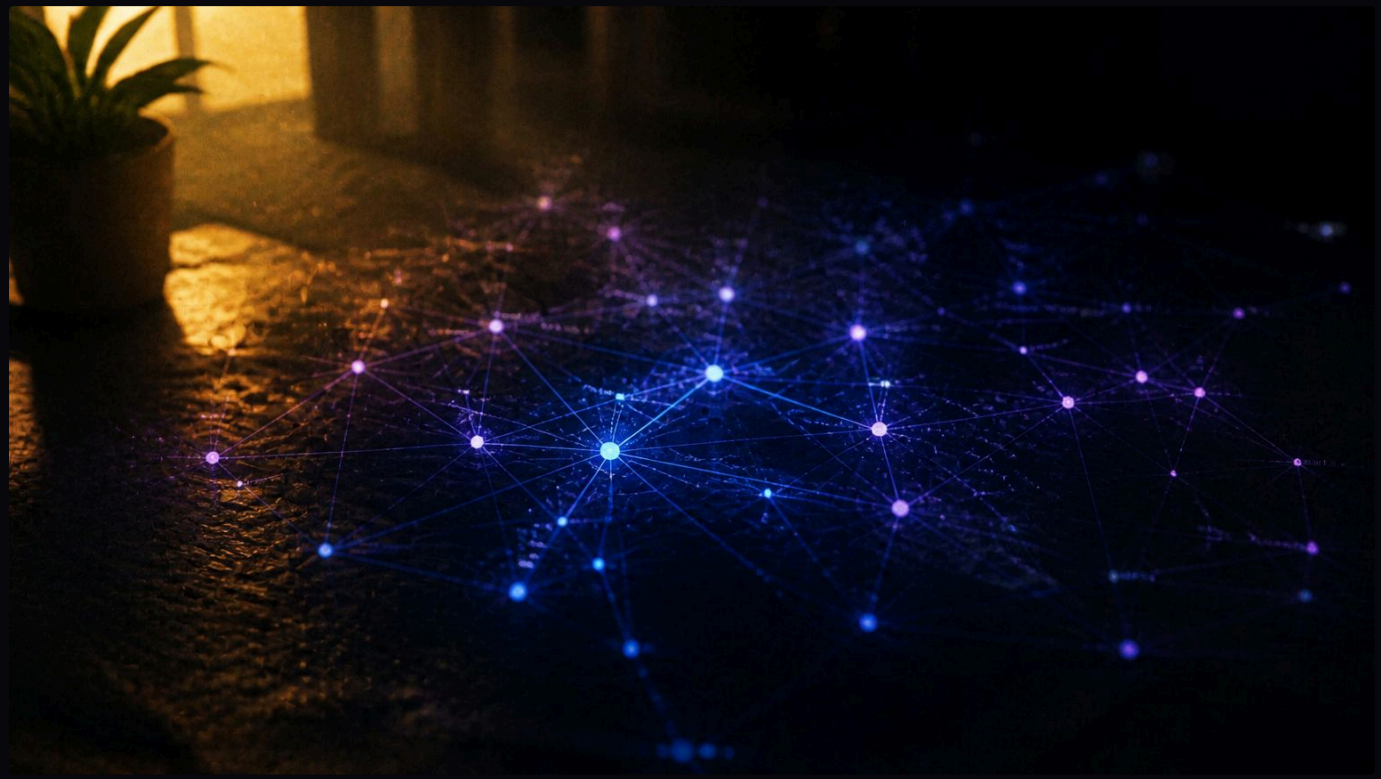
Combinados, MCP consulta la estructura exacta del grafo de Graphify, maximizando precisión y minimizando latencia.

HERRAMIENTA CLAVE

# MCP — codebase-memory-mcp

MCP permite que la IA interactúe con tu base de código como si fuera una herramienta externa, sin necesidad de inyectar el contenido completo en el contexto del prompt.

- Consulta directa de funciones, clases y módulos
- Ejecución remota de código sin cargar el archivo
- Reduce el consumo de tokens en **~95–98%**
- Tiempo de respuesta: **5–10 segundos**



HERRAMIENTA CLAVE

# Graphify

Graphify transforma tu proyecto en un grafo de conocimiento estructurado. La IA navega por él recuperando únicamente los nodos afectados por la consulta.

- Mapea relaciones entre archivos, funciones y dependencias
- Recupera solo los nodos relevantes para cada prompt
- Reduce el consumo de tokens en **~97–99%**
- Tiempo de respuesta: **3–6 segundos**

# La combinación ganadora: MCP + Graphify

### MCP consulta

La IA pide a MCP la estructura exacta que necesita del proyecto.

### Graphify responde

Devuelve únicamente los nodos del grafo afectados por la consulta.

### Resultado óptimo

**~98,5–99,5% de ahorro** en tokens y respuesta en **1–3 segundos**.

# Optimización en VibeCode: comparación de configuraciones

> Esta comparación está basada en un proyecto de código de referencia típico con aproximadamente 10.000 caracteres (2.500 tokens por archivo).

El impacto de usar MCP y Graphify es cuantificable. Esta tabla muestra la diferencia real en tiempo y tokens según la configuración utilizada.

| Configuración | Tiempo estimado | Consumo de tokens | % Ahorro | Descripción del proceso |
|---|---|---|---|---|
| **Sin nada instalado** | 15–30 s | ~100.000–150.000 | 0% (línea base) | La IA lee y procesa todo el contexto del archivo completo en cada prompt. |
| **Solo MCP** | 5–10 s | ~2.000–5.000 | ~95–98% | Consulta directa mediante herramientas o ejecución remota de código sin cargar todo el archivo. |
| **Solo Graphify** | 3–6 s | ~1.500–3.000 | ~97–99% | Navega por el grafo de conocimiento del proyecto y recupera solo los nodos afectados. |
| **MCP + Graphify** | 1–3 s | ~500–1.500 | ~98,5–99,5% | MCP consulta la estructura exacta del grafo de Graphify, reduciendo la latencia y maximizando la precisión. |

# Conclusiones clave

01

## Sé hiperespecífico en tus prompts

Indica archivo, función y acción concreta. Evita la ambigüedad para reducir alucinaciones.

02

## No cargues el proyecto entero

Usa MCP y Graphify para que la IA consulte solo lo necesario en lugar de leer archivos completos.

03

## Combina MCP + Graphify en VibeCode

La combinación reduce el consumo de tokens hasta un **99,5%** y el tiempo de respuesta a **1–3 segundos**.

# El Chef de la IA: Control y Eficiencia de Tokens

Cuatro herramientas esenciales para auditar modelos de IA generativa: cómo garantizar precisión, estructura y control sin improvisaciones **al trabajar en grandes cantidades de datos**.

GOBERNANZA DE IA | AUDITORÍA TÉCNICA

# El Problema: Un Genio que Improvisa

Los modelos de IA generativa son extraordinariamente creativos, pero tienen una debilidad crítica: **cuando no recuerdan un dato, lo inventan**. En auditoría, esto es inaceptable.

## Lo que hacen bien

Razonan, sintetizan y generan contenido complejo a partir de patrones aprendidos.

## Lo que no pueden garantizar

Exactitud factual, cálculos precisos y formato estructurado sin ayuda externa.

# La Metáfora del Restaurante

Para entender cómo se controla un modelo de IA en producción, imaginemos un **restaurante de alta cocina**. El modelo es el Chef principal: brillante, pero necesita un equipo que le garantice rigor.

## Advanced RAG

El Archivero

## Function Calling

La Calculadora

## Pydantic / JSON

El Molde

## Guardrails / Ragas

El Catador

HERRAMIENTA 1

# Advanced RAG — El Archivero

**Retrieval-Augmented Generation** asegura que el modelo nunca cocine sin ingredientes reales. Antes de responder, un sistema externo busca la información exacta en una base de datos verificada y se la entrega al modelo.

## Para el auditor

Permite trazar el origen de cada dato usado en la respuesta. Sin RAG, el modelo alucina; con RAG, cita fuentes.

HERRAMIENTA 2

# Function Calling — La Calculadora

Los modelos de IA no son fiables para operaciones matemáticas o llamadas a sistemas externos. Con **Function Calling**, el modelo delega esas tareas a funciones de código que ejecutan el cálculo exacto.


### El modelo detecta la necesidad

Identifica que debe calcular, consultar una API o ejecutar una acción.

### El código ejecuta la operación

Una función externa devuelve el resultado exacto, sin margen de error.

HERRAMIENTA 3

# Pydantic / JSON Schema — El Molde de Emplatado

La salida de un modelo de IA no puede ser libre: debe encajar en un **esquema estricto** para integrarse con sistemas downstream. Pydantic y JSON Schema definen la estructura, los tipos y las restricciones de cada campo.

## Validación de tipos

Garantiza que cada campo sea un número, fecha, cadena o enum según lo definido.

## Contrato de salida

El sistema receptor sabe exactamente qué esperar, eliminando errores de integración.

## Auditoría automática

Si la salida no cumple el esquema, se rechaza antes de procesarse.

HERRAMIENTA 4

# Guardrails / Ragas — El Catador de Calidad

Antes de que cualquier respuesta llegue al usuario final, un **sistema de guardrails** la inspecciona. Ragas proporciona métricas objetivas para evaluar la calidad de las respuestas generadas con RAG.

### Faithfulness

¿La respuesta se basa en los documentos recuperados?

### Answer Relevancy

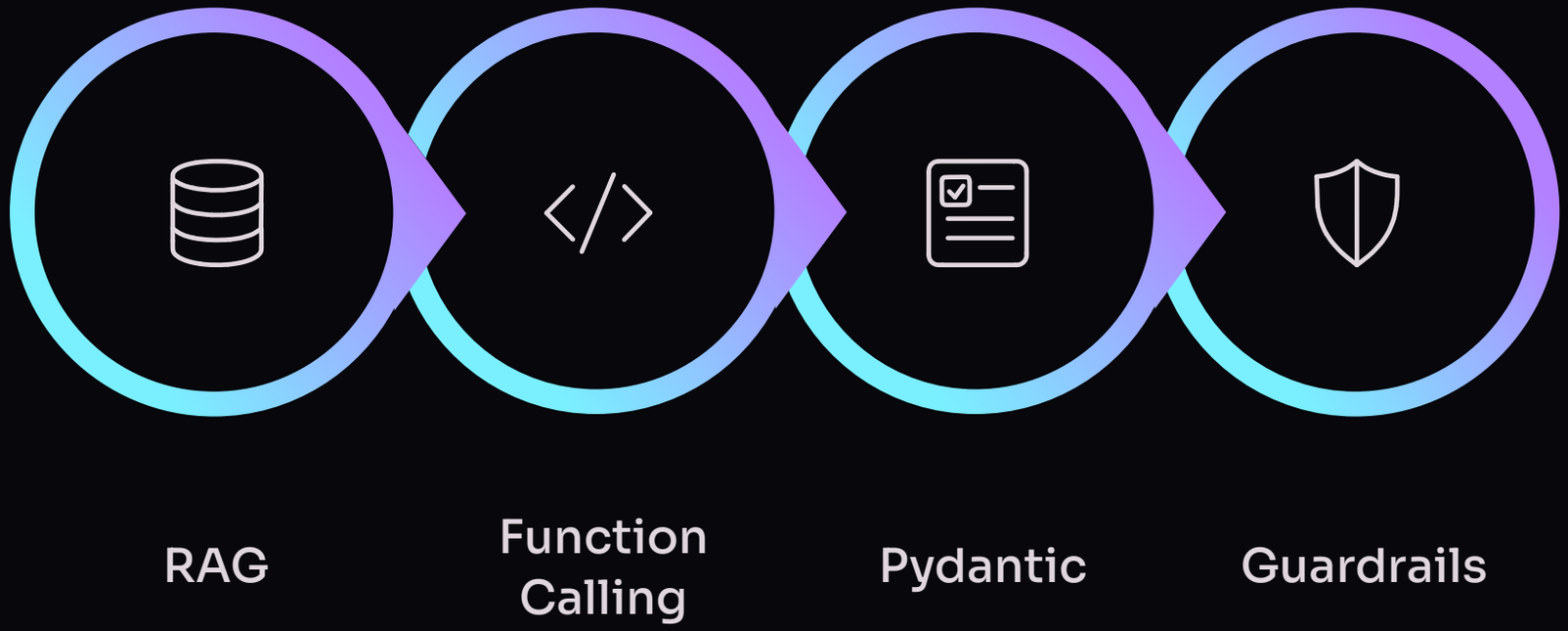
¿Responde realmente a la pregunta formulada?

### Context Precision

¿El contexto recuperado era el adecuado?

# Cómo Trabajan Juntas las Cuatro Herramientas

El flujo completo de un sistema de IA gobernado sigue una secuencia clara. Cada herramienta interviene en el momento preciso para eliminar un tipo de riesgo específico.




Este pipeline garantiza que cada respuesta sea **trazable, exacta, estructurada y validada** antes de llegar al usuario o sistema consumidor.

# Qué Debe Verificar el Auditor

Cada herramienta genera evidencias auditables. Estas son las preguntas clave que un auditor debe plantearse al evaluar un sistema de IA generativa.

1

### RAG — Trazabilidad del dato

¿Existe un registro de qué documentos se recuperaron para cada respuesta? ¿Se puede reproducir?

2

### Function Calling — Delegación verificada

¿Las funciones están documentadas? ¿Los resultados de las llamadas quedan registrados en logs?

3

### Pydantic — Contrato de salida

¿El esquema JSON está versionado? ¿Se rechazan respuestas que no lo cumplen?

4

### Guardrails — Métricas de calidad

¿Se monitorizan las métricas de Ragas en producción? ¿Hay umbrales definidos de aceptación?

# Conclusiones Clave

### RAG elimina alucinaciones

El modelo solo responde con datos recuperados y verificables.

### Function Calling garantiza exactitud

Los cálculos y acciones externas se delegan a código fiable.

### Pydantic asegura estructura

La salida siempre cumple un contrato definido y auditable.

### Guardrails cierran el ciclo

Ninguna respuesta sale sin pasar el control de calidad automatizado.

Un sistema de IA bien gobernado no elimina la creatividad del modelo: la **canaliza con rigor**, igual que un gran chef trabaja con ingredientes reales, herramientas precisas y estándares de calidad.