Ollama en Windows cuando falla

```
$ ollama serve
→ conexión a ollama.com:443
$ $env:OLLAMA_NO_CLOUD=1
→ 0 conexiones en 20 s
```

VRAM · 16 GB



Ollama en Windows cuando falla

La GPU que no se usa, el error 500, el contexto que se corta y las respuestas vacías: causa medida y arreglo comprobado

Edición: 04/11/2026. **Para quién:** usas Windows, ya instalaste Ollama (o vas a hacerlo) y algo no va como debería. Sabes abrir PowerShell y pegar un comando; no hace falta que programes.

Alcance de las cifras

Todo lo medido en este libro se midió el 04/11/2026 en un solo equipo: Windows 11 Pro, Intel Core Ultra 5 245K, 31 GB de RAM y una NVIDIA RTX 5060 Ti de 16 GB, con Ollama 0.40.14 en su versión zip portable. Las velocidades son de ese equipo; los síntomas, las líneas del registro y los arreglos te sirven en el tuyo. Cuando algo viene de la documentación oficial de Ollama y no de una medición nuestra, lo decimos en el mismo párrafo. Es el único aviso de este tipo que vas a leer.

Cómo usar este libro

Empieza por el capítulo 01 aunque tu problema sea otro: en menos de una hora aprendes a leer el registro de Ollama, que es donde está la causa de casi todo lo demás. Después ve directo a tu síntoma:

Lo que te pasa	Capítulo
Va lentísimo; el Administrador de tareas no muestra uso de la gráfica	02
Error 500 («Internal Server Error»), «unable to load model», el modelo no arranca	03
Responde como si no hubiera leído el documento entero; un agente hace cosas raras	04
La respuesta llega vacía, o «JSON no válido», tras un minuto de espera	05
La primera respuesta tarda mucho, la gráfica se queda llena, los juegos van a tirones	06
Quieres dejarlo bien configurado de una vez	07

Qué necesitas

- Ollama instalado con el instalador oficial o descomprimido desde el zip `ollama-windows-amd64.zip`.
- Al menos un modelo pequeño descargado. Los ejemplos usan `qwen3.5:4b` (3,3 GB): `ollama pull qwen3.5:4b`.
- PowerShell (viene con Windows). Si usas el zip, cambia `ollama` por la ruta completa, por ejemplo `C:\ollama\bin\ollama.exe`.
- Los scripts del paquete, en una carpeta como `C:\ollama-scripts`. Windows bloquea los scripts descargados; desbloquéalos una vez con `Get-ChildItem C:\ollama-scripts | Unblock-File` y lánzalos así: `powershell -ExecutionPolicy Bypass -File .\diagnostico.ps1`.

Contenido

Ollama en Windows cuando falla . . . 2
La GPU que no se usa, el error 500, el contexto que se corta y las respuestas vacías: causa medida y arreglo comprobado . . 2
Alcance de las cifras . . . 2
Cómo usar este libro . . . 2
Qué necesitas . . . 2
01 · Los primeros 60 minutos: el registro, la llamada a casa y tu contexto real 5
Paso 1 (10 minutos): encuentra el registro . . . 5
Paso 2 (20 minutos): la llamada a casa . . . 6
Paso 3 (25 minutos): cuánto texto ve tu modelo . . . 7
Paso 4 (5 minutos): una sospecha para más adelante . . . 8
Lo que te llevas de este capítulo . . . 8
02 · Ollama no usa la GPU 9
Cómo lo reconoces . . . 9
Primera pregunta: ¿Ollama ha encontrado tu gráfica? . . . 10
Causas cuando Ollama no ve la gráfica . . . 10
Causa cuando sí la ve pero el modelo no cabe . . . 11
No te fíes de un solo indicador . . . 11
Cómo compruebas que quedó arreglado . . . 11
Resumen . . . 12
03 · Error 500: el modelo no carga 13
Cómo lo reconoces . . . 13
La causa que medimos . . . 13
Arreglo paso a paso . . . 14
Antes de comprar hardware . . . 14
Cómo compruebas que quedó arreglado . . . 14
Resumen . . . 15
04 · El contexto que se corta sin avisar 16
Cómo lo reconoces . . . 16
Elige la cifra . . . 16
Dónde se pone: tres sitios y una trampa . . . 17
Cómo compruebas que quedó arreglado . . . 18
Resumen . . . 18
05 · Respuestas vacías y JSON roto 19
Síntoma 1: la respuesta vacía tras un minuto . . . 19
Síntoma 2: JSON que casi es JSON . . . 20
Resumen . . . 21

06 · Lento, con la gráfica llena o a tirones **22**

Síntoma 1: la primera respuesta tarda mucho . . . 22

Síntoma 2: la gráfica se queda llena después de usar Ollama . . . 22

Síntoma 3: el modelo iba rápido y ahora va lento . . . 23

Una trampa al medir: la caché de prompt . . . 24

Resumen . . . 24

07 · La configuración que funciona **25**

1. Las variables del servidor . . . 25

2. Un Modelfile por cada modelo que uses a diario . . . 25

3. Las opciones de cada petición . . . 26

4. La comprobación final . . . 26

5. Tu memoria de gráfica: qué modelo probar primero . . . 27

Resumen . . . 27

01 · Los primeros 60 minutos: el registro, la llamada a casa y tu contexto real

Antes de reinstalar, cambiar de modelo o comprar memoria, dedica una hora a tres comprobaciones. La primera te enseña dónde mira Ollama cuando algo falla. La segunda descubre algo que casi nadie cuenta: con la configuración por defecto, Ollama se conecta a internet nada más arrancar, aunque no le hayas pedido nada. La tercera te dice cuánto texto ve de verdad tu modelo, que en una gráfica de 16 GB resultó ser bastante menos de lo que parece.

No necesitas nada más que Ollama, un modelo pequeño descargado y PowerShell. Todo lo que cambies aquí se deshace en un minuto.

Paso 1 (10 minutos): encuentra el registro

Las aplicaciones que usan Ollama te enseñan mensajes cortos como «error», «no responde» o una respuesta vacía. Ollama, en cambio, escribe en su registro qué gráfica ha encontrado, cuánta memoria reserva, qué contexto usa y por qué ha fallado. Casi todos los problemas de este libro se entienden ahí.

Si instalaste Ollama con el instalador, la documentación oficial (docs.ollama.com/windows) indica que el registro del servidor está en `%LOCALAPPDATA%\Ollama\server.log`. Ábrelo así:

```
notepad "$env:LOCALAPPDATA\Ollama\server.log"
```

Si usas el zip portable, el registro va a la pantalla, así que hay que mandarlo a un archivo al arrancar. Con el zip descomprimido en `C:\ollama\bin`:

```
Get-Process ollama -ErrorAction SilentlyContinue | Stop-Process -Force
Start-Process C:\ollama\bin\ollama.exe -ArgumentList serve -WindowStyle Hidden `
  -RedirectStandardError C:\ollama\serve.log
```

Ahora busca estas cuatro líneas. Son las de nuestro equipo, copiadas tal cual (hemos quitado la fecha del principio):

```
msg="Ollama cloud disabled: false"
msg="Listening on 127.0.0.1:11434 (version 0.40.14)"
msg="inference compute" ... library=CUDA ... description="NVIDIA GeForce RTX 5060 Ti"
msg="vram-based default context" total_vram="15.9 GiB" default_num_ctx=4096
```

Qué te dice cada una:

- `cloud disabled: false`: la nube de Ollama está activa. Lo vemos en el paso 2.
- `Listening on`: dónde escucha y qué versión tienes. `127.0.0.1` significa que solo responde a tu propio equipo.
- `inference compute`: la gráfica que va a usar. Si esta línea no aparece o dice `library=cpu`, ve al capítulo 02.
- `default context ... default_num_ctx=4096`: cuánto texto ve el modelo si nadie le dice otra cosa. Lo vemos en el paso 3.

Para no leer el archivo entero, este comando saca solo esas líneas (cambia la ruta si usas el zip):

```
Select-String -Path "$env:LOCALAPPDATA\Ollama\server.log" `
  -Pattern 'cloud disabled','Listening on','inference compute','default context'
```

El paquete incluye `diagnostico.ps1`, que hace esto mismo y añade los errores, el reparto de memoria y los modelos cargados. Lo usarás en todos los capítulos.

Paso 2 (20 minutos): la llamada a casa

Lo que medimos

Arrancamos Ollama con su configuración por defecto y vigilamos durante 20 segundos las conexiones de red del proceso y de sus procesos hijos. No le pedimos nada. En esos 20 segundos abrió una conexión cifrada a `34.36.133.15`, puerto 443, cuyo nombre inverso es de `googleusercontent.com`: la dirección a la que resuelve ollama.com.

El registro explica por qué. Además de `cloud disabled: false`, aparece una tarea llamada `model recommendations cache`, que descarga de ollama.com una lista de modelos «recomendados» y la guarda en `%USERPROFILE%\.ollama\cache\model-recommendations.json`. En esa misma carpeta Ollama creó una pareja de claves, `id_ed25519` e `id_ed25519.pub`: la identidad de tu equipo ante ollama.com, que sirve para iniciar sesión o subir modelos.

La lista de recomendaciones traía modelos con el sufijo `:cloud`, y uno de ellos venía marcado con `required_plan: pro`. Un modelo `:cloud` se ejecuta en los servidores de Ollama, no en tu gráfica: lo que le escribas sale de tu equipo. Aparece en el mismo catálogo y se usa con los mismos comandos que un modelo local, así que es fácil elegir uno sin darse cuenta.

Después repetimos con la variable `OLLAMA_NO_CLOUD=1`. El registro pasó a decir `Ollama cloud disabled: true` y en los 20 segundos de observación no hubo ninguna conexión externa. Con la nube desactivada y `qwen3.5:4b` respondiendo, vigilamos 40 segundos más: ninguna conexión externa.

Compruébalo en tu equipo

Pega esto en PowerShell con Ollama arrancado. Vigila los procesos de Ollama durante 20 segundos, consultando cada 300 milisegundos, y apunta las conexiones que no son locales:

```
$fin = (Get-Date).AddSeconds(20); $vistas = @{}
while ((Get-Date) -lt $fin) {
  $ids = @(Get-Process -Name 'ollama','ollama app' -ErrorAction SilentlyContinue | ForEach-Object Id)
  $ids += @(Get-CimInstance Win32_Process | Where-Object { $ids -contains $_.ParentProcessId } |
    ForEach-Object ProcessId)
  Get-NetTCPConnection -ErrorAction SilentlyContinue |
    Where-Object { $ids -contains $_.OwningProcess -and
      "$($_.RemoteAddress)" -notin '0.0.0.0','127.0.0.1','::','::1' } |
    ForEach-Object { $vistas["$($_.RemoteAddress):$($_.RemotePort)"] = 1 }
  Start-Sleep -Milliseconds 300 }
if ($vistas.Count) { $vistas.Keys } else { '0 conexiones externas' }
```

Para ver la conexión del arranque, lanza el bloque y, sin esperar, arranca Ollama en otra ventana. El script `comprobar-red.ps1` del paquete hace lo mismo con la duración que le pidas y resuelve el nombre de cada dirección.

Sus límites, para que no le pidas más de lo que da: una conexión que se abra y se cierre entre dos consultas puede no aparecer; solo vigila a Ollama, no a la aplicación que uses encima; y te dice a dónde se conecta, no qué envía, porque va cifrado.

Córtalo

Con el zip portable, define la variable en la misma ventana antes de arrancar:

```
Get-Process ollama -ErrorAction SilentlyContinue | Stop-Process -Force
$env:OLLAMA_NO_CLOUD = '1'
Start-Process C:\ollama\bin\ollama.exe -ArgumentList serve -WindowStyle Hidden `
  -RedirectStandardError C:\ollama\serve.log
```

Con la aplicación instalada, la documentación oficial explica que Ollama hereda las variables de entorno de tu usuario de Windows. Créala, cierra Ollama desde su icono de la bandeja (clic derecho, salir) y ábrelo de nuevo desde el menú Inicio:

```
[Environment]::SetEnvironmentVariable('OLLAMA_NO_CLOUD', '1', 'User')
```

La misma documentación (docs.ollama.com/faq) da una alternativa: poner `"disable_ollama_cloud": true` en `%USERPROFILE%\.ollama\server.json` y reiniciar Ollama. Nosotros medimos la variable con el zip; el instalador no lo probamos, así que con la aplicación la comprobación de abajo es obligatoria.

Comprueba que quedó cortado

1. En el registro, la línea debe decir `Ollama cloud disabled: true`. Si sigue en `false`, Ollama no ha visto la variable: ciérralo del todo (también desde la bandeja) y vuelve a abrirlo.
2. Repite la vigilancia de 20 segundos durante el arranque: debe salir `0 conexiones externas`.
3. Haz una pregunta al modelo mientras vigilas: tampoco debe haber conexiones.
4. Ejecuta `ollama list` y comprueba que ningún modelo que uses termina en `:cloud`.

Un detalle práctico: descargar un modelo (`ollama pull`) necesita internet, claro. Descarga lo que vayas a usar y después trabaja con la nube desactivada.

Paso 3 (25 minutos): cuánto texto ve tu modelo

Lo que medimos

La cuarta línea del paso 1 es la que más problemas explica: `default_num_ctx=4096`. Con nuestra gráfica de 16 GB, Ollama decidió que el modelo trabajaría con 4.096 tokens de contexto. La documentación oficial (docs.ollama.com/context-length) confirma la regla: por debajo de 24 GiB de VRAM, 4k de contexto por defecto; de 24 a 48 GiB, 32k; a partir de 48 GiB, 256k. Casi todas las gráficas de consumo están en el primer tramo.

El contexto es todo lo que el modelo tiene delante a la vez: instrucciones, documentos, conversación y su propia respuesta. Un token es un trozo de palabra; en español, una página de texto normal ocupa varios cientos.

El caso que mejor lo enseña: conectamos OpenCode, un agente de programación, a un modelo local. Antes de llegar a tu encargo, OpenCode envía sus propias instrucciones y la descripción de sus herramientas. El registro de Ollama contó 22.919 tokens en esa primera petición, más de cinco veces los 4.096 disponibles. El modelo no llegó a ver lo que le habíamos pedido. No hubo ningún error: uno de los modelos se puso a planificar una tarea que nadie había pedido y el otro contestó que no tenía acceso a archivos. Con la configuración global de OpenCode vacía, el arranque bajó a 7.440 tokens, que tampoco caben en 4.096. El capítulo 04 cuenta cómo se arregló.

Esa es la trampa: cuando lo que envías no cabe, Ollama no te avisa en la respuesta. Simplemente trabaja con lo que le cabe.

Mide tu documento

Coge un documento que quieras usar con el modelo, guárdalo como texto (`.txt` o `.md`) y ejecuta esto. Le pide al modelo que lo lea con un contexto amplio y que escriba un solo token; lo que nos interesa es cuántos tokens ha leído:

```
$texto = Get-Content C:\docs\mi-documento.txt -Raw -Encoding UTF8
$cuerpo = @{ model = 'qwen3.5:4b'; prompt = $texto; stream = $false; think = $false
  keep_alive = 0; options = @{ num_ctx = 32768; num_predict = 1 } } | ConvertTo-Json
$r = Invoke-RestMethod http://127.0.0.1:11434/api/generate -Method Post `
  -Body ([Text.Encoding]::UTF8.GetBytes($cuerpo)) -ContentType 'application/json'
"Tokens del documento: $($r.prompt_eval_count)"
```

`-Encoding UTF8` es imprescindible en el PowerShell que trae Windows: sin él, las tildes llegan rotas al modelo.

Compara la cifra con el `default_num_ctx` de tu registro. Si tu documento, más tus instrucciones, más la respuesta que esperas, pasa de ahí, el modelo no lo está viendo entero. El script `contar-tokens.ps1` hace la comparación por ti con 4.096, 8.192, 16.384 y 32.768.

Hay otra forma de mirar el contexto de un modelo ya cargado. Según la documentación, `ollama ps` muestra una columna CONTEXT junto a PROCESSOR:

```
ollama ps
```

Lo que no hay que hacer todavía

No subas el contexto «a lo grande» por si acaso: ocupa memoria de la gráfica y, si te pasas, el modelo deja de caber y se vuelve lento (capítulo 02) o no carga (capítulo 03). El capítulo 04 explica cómo elegir la cifra y dónde ponerla para que la respeten todas tus aplicaciones.

Paso 4 (5 minutos): una sospecha para más adelante

Si alguna vez una aplicación te devuelve una respuesta vacía, o «JSON no válido», después de casi un minuto de espera, no es que el modelo se haya colgado. En el laboratorio, un modelo que «razona» antes de responder gastó todo su contexto pensando y entregó un texto vacío, otra vez sin ningún error. Apunta que existe: el capítulo 05 lo reproduce y lo arregla con una línea.

Lo que te llevas de este capítulo

- [ ] Sabes dónde está tu registro y qué dicen sus cuatro líneas clave.
- [ ] Sabes si tu Ollama se conecta a ollama.com al arrancar y lo has cortado con `OLLAMA_NO_CLOUD=1`.
- [ ] Has comprobado `cloud disabled: true` y 0 conexiones externas mientras responde.
- [ ] Ninguno de tus modelos termina en `:cloud`.
- [ ] Conoces tu contexto por defecto y cuántos tokens ocupa tu documento.

Si solo querías saber si Ollama es de verdad local, ya tienes la respuesta medida. Si además algo te falla, el resto del libro va por síntomas: cada capítulo empieza por cómo lo reconoces y termina con cómo compruebas que se ha arreglado.