# Descarga de la caché KV con NetApp

NetApp artificial intelligence solutions

NetApp
August 20, 2026

# Tabla de contenidos

# Descarga de la caché KV con NetApp

La primera oleada de inversión en IA empresarial se centró en desarrollar la capacidad de entrenamiento y ajuste fino de los modelos, no en su aplicación a gran escala. A medida que las organizaciones pasan de la fase experimental a la de producción, el centro de gravedad se desplaza hacia la inferencia en tiempo real: asistentes de búsqueda, chatbots, copilotos y flujos de trabajo de agentes con los que los usuarios interactúan continuamente. En estos entornos, la utilización de la GPU aumenta rápidamente, no porque cada solicitud ejecute un ciclo completo de entrenamiento, sino porque cada pregunta de seguimiento, cada nuevo giro en una conversación y cada usuario simultáneo añaden carga a la pila de inferencia.

# Introducción

Pronto se aprecia una tendencia: la GPU está realizando una cantidad sorprendente de trabajo repetitivo, volviendo a calcular la atención sobre tokens que ya ha procesado. Esa repetición no es solo un problema de ajuste, sino que se debe a la arquitectura de la memoria. La respuesta del sector es una estrategia de memoria por niveles basada en la caché KV.

## ¿Qué es KV Cache?

En pocas palabras, el almacenamiento en caché KV (clave-valor) es una técnica que se utiliza para optimizar la inferencia de los modelos de lenguaje a gran escala (LLM) mediante el almacenamiento de valores calculados previamente en una caché KV, de modo que no sea necesario volver a calcular dichos valores cada vez que se genere un nuevo token, lo que de otro modo sería necesario.

Nota: para obtener una descripción técnica más detallada, consulta "Descripción general del almacenamiento en caché KV de Hugging Face".

## ¿Qué es la descarga de caché KV?

La mayoría de los motores de inferencia almacenan la caché KV en la memoria de la GPU de forma predeterminada. Eso funciona bien hasta que la demanda aumenta. El tamaño de la caché KV varía linealmente con el tamaño del lote (el número de prompts procesados al mismo tiempo) y la longitud de la secuencia (la longitud de cada conversación o flujo de generación). A medida que se amplían las ventanas de contexto, los chats de varios turnos se vuelven comunes y más usuarios y agentes consumen servicios de inferencia, los requisitos de la caché KV pueden superar rápidamente la memoria disponible de la GPU. Cuando eso pasa, a menudo se eliminan entradas útiles de la caché y el motor debe volver a calcular la atención para los tokens que ya ha procesado.

La descarga de la caché KV resuelve esa limitación trasladando la caché KV de una solicitud procesada previamente a un destino externo fuera de la memoria de la GPU o del acelerador, como la RAM del sistema, un disco local o un almacenamiento compartido. Las entradas descargadas pueden conservarse mientras la capacidad lo permita y volver a consultarse cuando llegue un prefijo similar o una solicitud de seguimiento, en lugar de descartarse cuando se llene la memoria de la GPU. En la práctica, estos destinos de descarga funcionan como un espacio de intercambio por niveles para la memoria de la GPU: más lento que la VRAM, pero más amplio y duradero, lo que amplía la cantidad de contexto conversacional y la carga de trabajo simultánea que el sistema puede soportar sin necesidad de realizar repetidamente tareas de precarga.

# Importancia de un nivel de almacenamiento compartido

La descarga de la caché KV ya resulta útil cuando un único motor de inferencia supera la capacidad de la memoria de la GPU. Trasladar bloques de caché a la RAM de la CPU amplía la capacidad en un servidor. Eso es útil, pero solo resuelve una parte del problema en producción. Las plataformas de inferencia reales rara vez se ejecutan como una única instancia de motor de servicio. Se ejecutan detrás de equilibradores de carga, se escalan horizontalmente y atienden a muchos usuarios simultáneos y agentes. En ese mundo, el almacenamiento compartido es lo que convierte la caché KV de una optimización local en una capacidad de la plataforma.

- **El almacenamiento compartido permite la reutilización entre instancias de servicio** Una de las principales ventajas del almacenamiento compartido es la capacidad de acceder a los mismos archivos u objetos desde múltiples instancias y nodos. Esto resulta especialmente relevante en el caso de una implementación escalada con dos o más instancias del motor de servicio detrás de un equilibrador de carga, cada una de ellas configurada para descargar bloques de caché KV al mismo depósito compartido o volumen NAS. Por ejemplo, cuando una instancia procesa un prefijo de consulta y descarga los bloques de caché resultantes, otra instancia puede cargar esos bloques al producirse una coincidencia en la caché, en lugar de volver a calcular el mismo trabajo de precarga. Esto es importante porque el tráfico de producción está distribuido, la primera pregunta de un usuario podría llegar a la instancia A; una pregunta de seguimiento u otro usuario con una consulta similar podría llegar a la instancia B. Sin almacenamiento compartido, cada instancia mantiene su propio entorno de caché privado.
- **La memoria de la CPU por sí sola no es suficiente para implementaciones con múltiples instancias** El nivel de CPU es rápido, pero es local para cada proceso del motor de servicio. Una instancia no puede acceder a las entradas de la caché KV que otra instancia ha descargado en la RAM local de su CPU, a menos que implementes un canal punto a punto, lo que introduce latencia adicional y complejidad operativa. El almacenamiento compartido elimina esa barrera. La memoria RAM de la CPU sigue siendo valiosa como un nivel de almacenamiento temporal rápido en cada nodo, pero el S3 o NAS compartido proporciona el plano de memoria común en el que varios motores pueden leer y escribir. Ya no escalas las GPU de forma aislada, ahora escalas con una infraestructura de caché compartida.
- **Implementación de una estrategia de diseño de tres niveles** + Una arquitectura preferible combina:

  - **Memoria de la GPU** — caché activa para las solicitudes en curso.
  - **Memoria de la CPU** — almacenamiento temporal rápido a medida que las solicitudes se incorporan a la cola.
  - **Almacenamiento compartido** — un almacén de datos duradero, de gran capacidad y que se puede compartir.

  Con esta arquitectura, los bloques de la caché KV pueden transferirse desde el nivel compartido a la memoria de la CPU a medida que llegan nuevas solicitudes, lo que permite que la GPU se enfoque en el trabajo activo sin dejar de mantener una caché duradera en el almacenamiento.

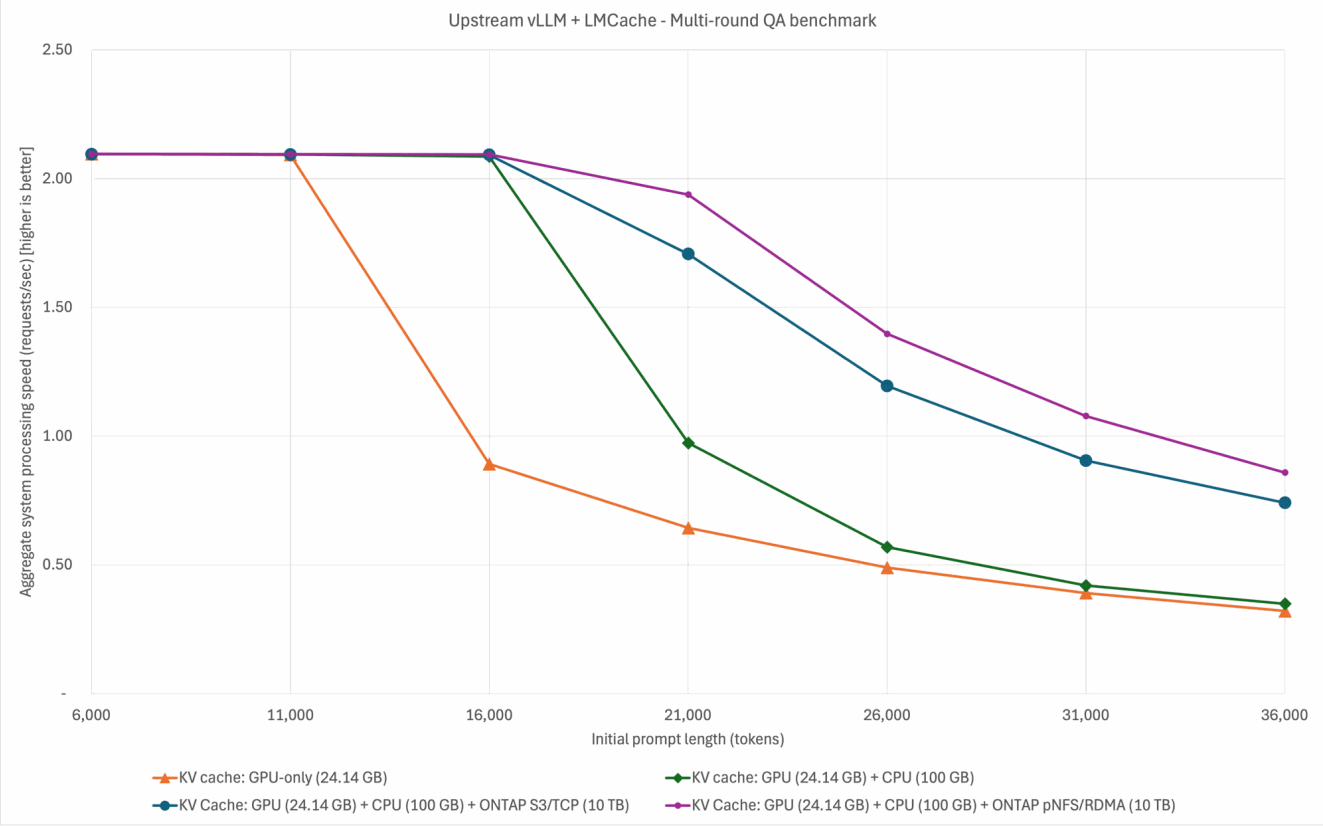
- **La economía de la inferencia, no solo la velocidad** Desde el punto de vista de la producción, la importancia del almacenamiento compartido no radica únicamente en la latencia. Se trata del control sobre la memoria, la concurrencia y el coste. Los motores de servicio como vLLM gestionan la caché KV de forma eficiente dentro de un mismo nodo. Los marcos de descarga de caché KV, como LMCache, convierten la caché KV en un activo portátil y compartible que puede desplazarse entre la memoria de la CPU y el almacenamiento. Las plataformas de almacenamiento compartido, como NetApp ONTAP y StorageGRID, proporcionan el nivel de durabilidad que hace que ese uso compartido sea viable entre múltiples instancias. Con LMCache conectado al almacenamiento compartido, múltiples instancias de vLLM pueden acceder a las mismas entradas de la caché KV descargada, lo que mejora el rendimiento y reduce la redundancia a medida que aumenta la concurrencia. Esto reduce el desperdicio de ciclos de GPU en el rellenado previo repetido, lo cual es especialmente relevante para chatbots, asistentes de

búsqueda, agentes y cualquier carga de trabajo con hilos de múltiples turnos, indicaciones compartidas del sistema o patrones de contexto recurrentes.

- **Mejora el rendimiento de inferencia y la inversión en GPU** + Este benchmark compara el rendimiento de inferencia a medida que aumentan la longitud de la conversación y la carga de usuarios simultáneos. Cuando la caché KV se mantiene solo en la memoria de la GPU, el margen disponible se agota rápidamente y el rendimiento cae (línea naranja). Con un diseño de tres niveles (GPU para el trabajo activo, CPU para una preparación rápida y almacenamiento compartido para una caché duradera y compartible), la plataforma mantiene más sesiones y evita esa misma caída pronunciada del rendimiento. En la práctica, la organización en niveles te ayuda a sacar más trabajo de las mismas GPU al reducir los cálculos repetidos de precarga.

  - **En pocas palabras:**
    - **Nivel de GPU** — maneja el trabajo activo y en curso.
    - **Nivel de CPU** — mantiene la caché utilizada recientemente cerca del motor de servicio.
    - **Nivel de almacenamiento compartido** — mantiene la caché en todos los servidores y libera memoria de la GPU y la CPU para nuevas sesiones.

Figura 1 - Prueba de rendimiento de inferencia en múltiples rondas




La prueba de rendimiento indica que añadir almacenamiento compartido junto con la memoria de la CPU no reduce el rendimiento; amplía el rango operativo útil antes de que aparezca una degradación del rendimiento. La arquitectura de almacenamiento por niveles añade de forma eficaz capas de memoria para la inferencia, reduciendo la necesidad de añadir GPU cada vez que aumentan el número de sesiones, la longitud del contexto o la concurrencia.

- **Adecuado para empresas: protocolos estándar, sin cliente propietario** + El almacenamiento compartido es importante, pero no todos los sistemas de almacenamiento compartido son iguales. NetApp admite la descarga de caché KV mediante protocolos y herramientas estándar del sector:

- **StorageGRID S3**
- **NFS / pNFS para ONTAP NAS**

No se requiere ningún cliente de inferencia propio y personalizado. Los equipos de operaciones pueden usar los mismos protocolos de almacenamiento que ya operan para otros servicios de datos, con la protección de datos, la seguridad y la movilidad multicloud detrás de ellos.

# Implementa la descarga de la caché KV

El almacenamiento compartido amplía la capacidad de la caché KV y permite su reutilización entre las instancias de servicio. Para utilizar ese nivel en producción, configuras un motor de inferencia y un framework de descarga de caché KV. Las siguientes secciones describen cómo implementar esta pila usando opciones de herramientas comunes.

## vLLM

vLLM es un popular motor de inferencia LLM de código abierto y alto rendimiento. En esta solución, es el motor el que recibe las solicitudes de los usuarios, genera las respuestas y gestiona la caché KV dentro de la memoria de la GPU durante la inferencia. vLLM es modular: puedes elegir qué framework de offloading deseas utilizar. Las siguientes secciones describen cómo implementar una pila de servicio basada en vLLM con frameworks de offloading populares.

## LMCache (modo en proceso)

LMCache es un popular marco de código abierto para la descarga de caché KV. En esta sección se describe cómo implementar una pila de servicio basada en vLLM que utiliza LMCache para la descarga de caché KV. En esta implementación, LMCache trabaja con vLLM para mover la caché KV fuera de la memoria de la GPU hacia niveles más grandes como la RAM de la CPU, el disco local o el almacenamiento compartido (como StorageGRID S3 o un montaje NAS de ONTAP).

En una configuración predeterminada, vLLM almacena la caché KV únicamente en la memoria de la GPU. A medida que aumenta la carga del sistema, eso se convierte en el cuello de botella. En esta solución, vLLM se combina con LMCache, donde vLLM sirve el modelo, mientras que LMCache descarga y reutiliza la caché entre la CPU y el almacenamiento compartido NetApp. LMCache se conecta a vLLM a través de un conector; lo habilitas al inicio con el flag --kv-transfer-config de vLLM, así vLLM y LMCache guardan la caché en el almacenamiento y la recargan cuando hay un acierto de caché.

El modo «in-process» es el método original para ejecutar LMCache con vLLM. Cuando usas el modo «in-process», LMCache se ejecuta dentro del proceso de vLLM. Las versiones posteriores de LMCache añadieron el modo multiproceso, que es el enfoque recomendado actualmente para una mejor compatibilidad de funciones y mayor rendimiento. Esta sección cubre el modo «in-process», que está marcado como obsoleto en la documentación actual de LMCache. Para nuevas implementaciones, considera usar el modo multiproceso en su lugar.

Para esta implementación, vas a:

- Instala vLLM junto con LMCache
- Inicia vLLM con el conector LMCache activado
- Implementa una configuración de tres niveles (GPU + CPU + almacenamiento compartido) con dos instancias de vLLM (en GPU independientes) detrás de un enrutador vLLM, ambas utilizando el mismo backend de almacenamiento compartido.

## Prerrequisitos

- Servidor Linux con GPU y controladores NVIDIA (incluido CUDA) y al menos una GPU (dos GPU o varios servidores para la configuración completa de dos instancias + router)
- Python y uv instalados
- Docker y NVIDIA Container Toolkit instalados (requeridos para el enrutador vLLM en el paso 4)
- Acceso a la red para descargar el modelo (por ejemplo, Hugging Face) y para conectarte a tu endpoint de almacenamiento

#### Backend de StorageGRID S3

- Depósito de S3 creado para uso como caché KV
- Credenciales de lectura/escritura para el bucket
- Conectividad de red desde el servidor de la GPU hasta el endpoint de S3
- Solicitudes S3 de tipo alojado virtualmente habilitadas

#### Backend pNFS/NFS de ONTAP

- Volumen de ONTAP exportado a los servidores de GPU
- Ruta de montaje creada (por ejemplo, /mnt/kvcache) en **cada** servidor que ejecuta vLLM. Ejemplo de comando de montaje para pNFS de alto rendimiento sobre RDMA (RoCE):

```
mount -o
vers=4.1,proto=rdma,trunkdiscovery,write=eager,rsize=262144,wsize=262144
my_storage.my_company.net:/kvcache /mnt/kvcache
```

[[1-install-vllm-and-lmcache]]
=== 1. Instala vLLM y LMCache

Crea un entorno virtual e instala vLLM y LMCache. Consulta la LMCache "documentación de instalación" para obtener más detalles. Es fundamental que sigas la ruta de instalación correcta para tu versión de CUDA.

```
uv venv .venv
source .venv/bin/activate
uv pip install --upgrade lmcache vllm
```

En este momento ya dispones del motor de inferencia (vLLM) y de la capa de caché (LMCache).

[[2-configure-lmcache]]
=== 2. Configura LMCache

vLLM no descarga la caché KV por sí mismo. Tú habilitas LMCache a través del conector de transferencia KV de vLLM. Crea un archivo YAML (lmcache-config.yaml) que defina la configuración de la CPU y el nivel de almacenamiento compartido. LMCache lee este archivo YAML como parte de la secuencia de inicio de la inferencia de vLLM.

**Fragmento de código de ejemplo: nivel compartido de StorageGRID S3**

```
local_cpu: True
max_local_cpu_size: 100
save_decode_cache: True
remote_url: "s3://<bucket>.<storagegrid_s3_endpoint_fqdn>"
remote_serde: "naive"
extra_config:
  s3_num_io_threads: 320
  s3_prefer_http2: False
  s3_enable_s3express: False
  save_chunk_meta: False
  disable_tls: <True/False>
  aws_access_key_id: "<storagegrid_s3_access_key>"
  aws_secret_access_key: "<storagegrid_s3_secret_key>"
  s3_region: "us-east-1"
```

Nota: Sustituye el nombre del bucket, el punto de conexión, las credenciales y `disable_tls` para que se ajusten a tu entorno.

**Fragmento de ejemplo: ONTAP pNFS/NAS nivel compartido**

```
local_cpu: true
max_local_cpu_size: 100
save_decode_cache: true
remote_storage_plugins: ["fs"]
extra_config:
  remote_storage_plugin.fs.base_path: /mnt/kvcache
  save_chunk_meta: false
```

Monta la exportación de ONTAP usando el procedimiento NFS/pNFS de tu sitio antes del paso 3.

[[3-run-two-vllm-instances-shared-cache-across-servers]]
=== 3. Ejecuta dos instancias de vLLM (caché compartida entre servidores)

Configura las variables de entorno comunes en ambas instancias:

```
export LMCACHE_CONFIG_FILE=$(pwd)/lmcache-config.yaml
export PYTHONHASHSEED=0
export VLLM_API_KEY='<shared_api_key>'
```

**Instancia 1** (GPU 0, puerto 8001):

```
export CUDA_VISIBLE_DEVICES=0
```

```
vllm serve Qwen/Qwen3-8B \
  --tensor-parallel-size 1 \
  --kv-transfer-config
'{"kv_connector":"LMCacheConnectorV1","kv_role":"kv_both"}' \
  --port 8001
```

**Instancia 2** (GPU 1, puerto 8002 — terminal independiente):

```
export CUDA_VISIBLE_DEVICES=1
vllm serve Qwen/Qwen3-8B \
  --tensor-parallel-size 1 \
  --kv-transfer-config
'{"kv_connector":"LMCacheConnectorV1","kv_role":"kv_both"}' \
  --port 8002
```

Utiliza el mismo archivo de configuración y la misma semilla de hash para que ambas instancias coincidan en la identidad de los bloques de caché y puedan leer los datos descargados de la otra desde el almacenamiento compartido. Configura `VLLM_API_KEY` en ambas instancias; el enrutador pasa este valor como `OPENAI_API_KEY` para que las instancias acepten las peticiones enrutadas.

Nota: También se puede utilizar una única instancia de GPU para implementar una arquitectura de almacenamiento por niveles. En tal caso omite el paso 4.

[[4-deploy-the-vllm-router-single-entry-point]]
=== 4. Implementa el enrutador vLLM (único punto de entrada)

Después de que ambas instancias estén operativas, implementa un enrutador para que los clientes usen una única URL compatible con OpenAI.

```
docker run -d --rm --name vllm-router \
  --network host \
  -e OPENAI_API_KEY="${VLLM_API_KEY}" \
  ghcr.io/vllm-project/production-stack/router:latest \
  --port 8000 \
  --service-discovery static \
  --static-backends "http://localhost:8001,http://localhost:8002" \
  --static-models "Qwen/Qwen3-8B,Qwen/Qwen3-8B" \
  --routing-logic roundrobin
```

Envía tráfico a http://localhost:8000/v1. El router distribuye las solicitudes; LMCache y el almacenamiento compartido permiten reutilizar la caché entre instancias, no solo dentro de una misma GPU.

Por ejemplo, puedes enviar una solicitud al router utilizando curl de la siguiente manera (sustituye <shared_api_key> por tu clave de API):

```
curl -s http://localhost:8000/v1/chat/completions \
```

```
  -H "Content-Type: application/json" \
  -H 'Authorization: Bearer <shared_api_key>' \
  -d '{
    "model": "Qwen/Qwen3-8B",
    "messages": [{"role": "user", "content": "What is 2+2?"}]
  }' | jq .
```

[[5-validate-that-tiering-is-working]]
=== 5. Comprueba que la organización por niveles funciona

- Ambos procesos vLLM escuchan en 8001 y 8002; el router responde en 8000.
- Envía una solicitud con un prefijo largo a través del router.
- Confirma que los objetos o archivos aparezcan en el almacenamiento compartido (S3 bucket o ls -ltr /mnt/kvcache).
- Envía de nuevo una solicitud similar: la segunda solicitud debería mostrar un rellenado previo más rápido o un comportamiento de acierto en caché en los registros.
- Repite a través del router para que el tráfico pueda llegar a la otra instancia.

# Conclusión

La implementación de una arquitectura de almacenamiento por niveles traslada la caché KV de la memoria de la GPU a la memoria de la CPU y al almacenamiento compartido NetApp, así la inferencia puede escalar con los usuarios y la longitud del contexto sin desperdiciar ciclos de GPU en rellenados previos repetidos. El almacenamiento compartido convierte la caché en una infraestructura reutilizable entre las instancias de servicio y te ayuda a sacar más valor de tu inversión actual en GPU.

NetApp storage es una opción ideal para este nivel porque ofrece lo que la inferencia en producción necesita más allá de la capacidad bruta: acceso estándar a través de S3 y NFS/pNFS, caché compartida que varias instancias del motor de servicio pueden usar al mismo tiempo, y servicios de datos empresariales en los que ya confías para la durabilidad, protección y gobernanza. No necesitas un cliente propietario; los frameworks de descarga de caché KV se conectan a StorageGRID S3 o a un montaje NAS de ONTAP usando protocolos conocidos.

NetApp te ofrece una base de almacenamiento conocida para la descarga de la caché KV sin que tengas que cambiar cómo ejecutas la inferencia ni cómo tus equipos gestionan los datos.