Scarico della KV cache con NetApp

NetApp artificial intelligence solutions

NetApp
August 20, 2026

Sommario

Scarico della KV cache con NetApp . . . 1
- Introduzione . . . 1
  - Cos’è la cache KV? . . . 1
  - Che cos’è l’offloading della cache KV? . . . 1
- Importanza di un tier di storage condiviso . . . 2
- Distribuisci l’offload della cache KV . . . 4
  - vLLM . . . 4
  - LMCache (modalità in-process) . . . 4
  - Prerequisiti . . . 4
- Conclusione . . . 8

Scarico della KV cache con NetApp

La prima ondata di investimenti nell'AI aziendale si è concentrata sull'addestramento dei modelli e sulla messa a punto delle capacità di sviluppo, non sulla loro implementazione su larga scala. Man mano che le organizzazioni passano dalla fase di sperimentazione a quella di produzione, il centro di gravità si sposta sull'inferenza in tempo reale: assistenti di ricerca, chatbot, copiloti e flussi di lavoro basati su agenti con cui interagisci continuamente. In questi ambienti, l'utilizzo della GPU aumenta rapidamente non perché ogni richiesta esegua un ciclo di addestramento completo, ma perché ogni domanda di approfondimento, ogni nuovo turno di conversazione e ogni utente simultaneo aggiunge carico allo stack di inferenza.

Introduzione

Ben presto emerge uno schema: la GPU esegue una quantità sorprendente di lavoro ripetuto, ricalcolando l'attenzione su token già elaborati. Questa ripetizione non è solo un problema di ottimizzazione; è un problema di architettura della memoria. La risposta del settore è una strategia di memoria a livelli basata sulla KV cache.

Cos'è la cache KV?

In parole semplici, la cache KV (key-value) è una tecnica utilizzata per ottimizzare l'inferenza dei large language model (LLM) memorizzando i valori calcolati in precedenza in una cache KV, così questi valori non devono essere ricalcolati ogni volta che viene generato un nuovo token, cosa che altrimenti sarebbe necessaria.

Nota: per una panoramica tecnica più approfondita, consulta "Panoramica sulla cache KV di Hugging Face".

Che cos'è l'offloading della cache KV?

La maggior parte dei motori di inferenza memorizza la KV cache nella memoria GPU per impostazione predefinita. Questo funziona bene finché la domanda non aumenta. La dimensione della KV cache scala linearmente con la dimensione del batch (il numero di prompt elaborati contemporaneamente) e la lunghezza della sequenza (la lunghezza di ogni conversazione o flusso di generazione). Con l'espansione delle finestre di contesto, la diffusione delle chat multi-turno e l'aumento del numero di utenti e agenti che utilizzano i servizi di inferenza, i requisiti della KV cache possono superare rapidamente la memoria GPU disponibile. In tal caso, le voci utili della KV cache vengono spesso eliminate e il motore deve ricalcolare l'attenzione per i token già elaborati.

Lo scaricamento della KV cache risolve questo problema spostando la KV cache di un prompt già elaborato verso una destinazione esterna alla memoria della GPU o dell'acceleratore, come la RAM di sistema, il disco locale o lo storage condiviso. Le voci scaricate possono essere conservate finché la capacità lo consente e richiamate nuovamente quando arriva un prefisso o una richiesta successiva simile, invece di essere scartate quando la memoria della GPU si riempie. In pratica, queste destinazioni di scaricamento funzionano come uno spazio di swap tiered per la memoria della GPU: più lento della VRAM, ma più ampio e resistente, estendendo la quantità di contesto conversazionale e carico di lavoro simultaneo che il sistema può sostenere senza ripetute operazioni di prefill.

Importanza di un tier di storage condiviso

L'offload della KV cache è già utile quando un singolo motore di inferenza supera la memoria della GPU. Spostare i blocchi di cache nella RAM della CPU estende la capacità su un server. Questo è utile, ma risolve solo una parte del problema in produzione. Le piattaforme di inferenza reali raramente funzionano come una singola istanza del motore di inferenza. Vengono eseguite dietro bilanciatori di carico, scalano orizzontalmente e servono molti utenti simultanei e agenti. In questo contesto, lo storage condiviso è ciò che trasforma la KV cache da un'ottimizzazione locale in una funzionalità di piattaforma.

- **L'archiviazione condivisa consente il riutilizzo tra istanze di servizio** Uno dei principali vantaggi dell'archiviazione condivisa è la possibilità di accedere agli stessi file o oggetti su più istanze e nodi. Questo è particolarmente vero nel caso di una distribuzione scalata di due o più istanze del motore di servizio dietro un bilanciatore di carico, ciascuna configurata per scaricare i blocchi della cache KV nello stesso bucket condiviso o volume NAS. Ad esempio, quando un'istanza elabora un prefisso di prompt e scarica i blocchi di cache risultanti, un'altra istanza può caricare quei blocchi in caso di cache hit invece di ricalcolare lo stesso lavoro di prefill. Questo è importante perché il traffico di produzione è distribuito: la prima domanda di un utente potrebbe arrivare all'istanza A; una domanda successiva o un altro utente con un prompt di sistema simile potrebbe arrivare all'istanza B. Senza archiviazione condivisa, ogni istanza mantiene il proprio mondo di cache privato.
- **La sola memoria della CPU non è sufficiente per le implementazioni multi-istanza.** Il tier CPU è veloce, ma è locale a ciascun processo del motore di servizio. Un'istanza non può accedere alle voci della cache KV che un'altra istanza ha scaricato nella sua RAM locale della CPU, a meno che tu non implementi un canale peer-to-peer, il che introduce ulteriore latenza e complessità operativa. Lo storage condiviso elimina questo ostacolo. La RAM della CPU rimane preziosa come tier di staging veloce su ciascun nodo, ma S3 o NAS condivisi forniscono il piano di memoria comune che più motori possono leggere e scrivere. Non stai più scalando le GPU in isolamento, stai scalando con un'infrastruttura di cache condivisa.
- **Abilitare una strategia di progettazione a tre livelli** + Una architettura preferibile combina:
    - **Memoria GPU** — cache attiva per le richieste in corso.
    - **Memoria della CPU** — staging rapido mentre i prompt entrano nella coda.
    - **Archiviazione condivisa** — supporto di archiviazione durevole, capiente e condivisibile.

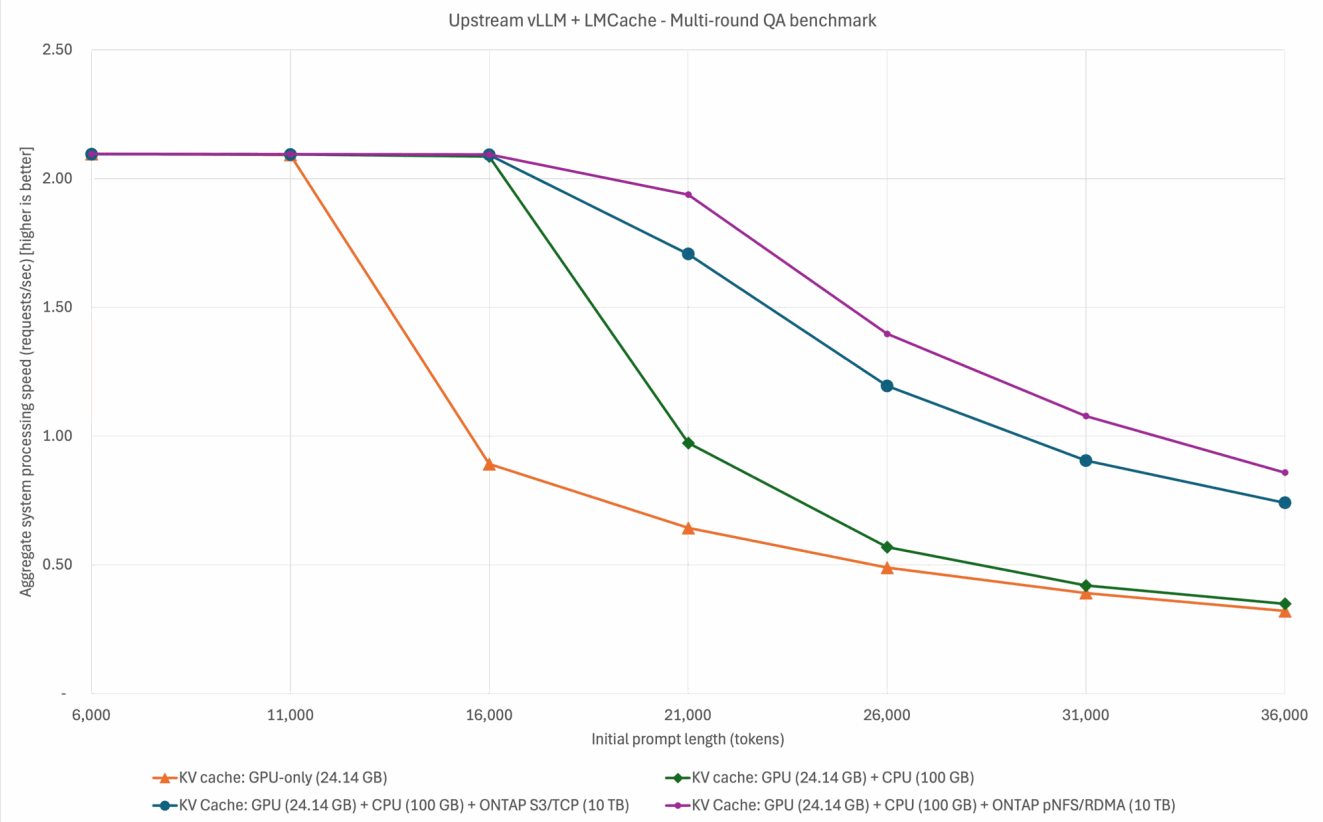
  Con questa architettura, i blocchi della cache KV possono essere trasferiti dal tier condiviso alla memoria della CPU man mano che arrivano nuove richieste, mantenendo la GPU concentrata sul lavoro attivo e conservando comunque una cache durevole sullo storage.

- **Economia dell'inferenza, non solo velocità** Dal punto di vista della produzione, l'importanza dello storage condiviso non è solo la latenza. È il controllo sulla memoria, la concorrenza e i costi. I motori di servizio come vLLM gestiscono la KV cache in modo efficiente all'interno di un singolo nodo. I framework di offload della KV cache come LMCache trasformano la KV cache in una risorsa portatile e condivisibile che può spostarsi tra la memoria della CPU e lo storage. Le piattaforme di storage condiviso come NetApp ONTAP e StorageGRID forniscono il tier durevole che rende praticabile questa condivisione tra più istanze. Con LMCache collegato allo storage condiviso, più istanze vLLM possono accedere alle stesse voci della KV cache offloadata, migliorando il throughput e riducendo i calcoli ridondanti man mano che la concorrenza cresce. Questo riduce i cicli GPU sprecati per il prefill ripetuto, un aspetto direttamente rilevante per chatbot, assistenti di ricerca, agenti e qualsiasi workload con thread multi-turno, prompt di sistema condivisi o pattern di contesto ricorrenti.
- **Migliora le prestazioni di inferenza e l'investimento in GPU** + Questo benchmark confronta le prestazioni di inferenza all'aumentare della durata della conversazione e del carico di utenti simultanei. Quando la cache KV è mantenuta solo nella memoria GPU, lo spazio disponibile si esaurisce rapidamente e il throughput diminuisce (linea arancione). Con un'architettura a tre livelli (GPU per il lavoro attivo, CPU per lo staging veloce e storage condiviso per una cache durevole e condivisibile), la piattaforma supporta

più sessioni ed evita lo stesso brusco calo delle prestazioni. In pratica, la suddivisione in livelli ti aiuta a ottenere più lavoro dalle stesse GPU riducendo i calcoli di prefill ripetuti.

- **In parole semplici:**
  - **Livello GPU** — gestisce il lavoro attivo in corso.
  - **Livello CPU** — mantiene la cache usata di recente vicino al motore di servizio.
  - **Shared storage tier** — preserva la cache tra i server e libera la memoria GPU/CPU per nuove sessioni.

Figura 1 - Benchmark di inferenza multi-round



Il benchmark indica che l'aggiunta di storage condiviso alla memoria della CPU non riduce le prestazioni; estende l'intervallo operativo utilizzabile prima che si verifichi un degrado del throughput. Il tiering aggiunge di fatto strati di memoria per l'inferenza, riducendo la necessità di aggiungere GPU ogni volta che aumentano il numero di sessioni, la lunghezza del contesto o la concorrenza.

- **Ideale per le aziende: protocolli standard, nessun client proprietario** + Lo storage condiviso è importante, ma non tutti gli storage condivisi sono uguali. NetApp supporta lo scaricamento della KV cache utilizzando protocolli e strumenti standard del settore:
  - **StorageGRID S3**
  - **NFS / pNFS per ONTAP NAS**

Non è necessario alcun client di inferenza proprietario personalizzato. I team operativi possono utilizzare gli stessi protocolli di archiviazione che già usano per altri servizi dati, con la consueta protezione dei dati, sicurezza e mobilità multicloud alle spalle.

Distribuisci l’offload della cache KV

Lo storage condiviso estende la capacità della cache KV e ne consente il riutilizzo tra le istanze di servizio. Per usare quel tier in produzione, configuri un inference engine e un framework di offload della cache KV. Le sezioni seguenti descrivono come implementare questo stack usando le opzioni di strumenti più comuni.

vLLM

vLLM è un popolare motore di inferenza LLM open-source dalle performance elevate. In questa soluzione, è il motore che riceve le richieste degli utenti, genera le risposte e gestisce la KV cache all’interno della memoria GPU durante l’inferenza. vLLM è pluggable: puoi scegliere quale framework di offloading vuoi usare. Le sezioni seguenti descrivono come implementare uno stack di serving basato su vLLM con i framework di offloading più diffusi.

LMCache (modalità in-process)

LMCache è un popolare framework open-source per l’offload della cache KV. Questa sezione descrive come implementare uno stack di serving basato su vLLM che utilizza LMCache per l’offload della cache KV. In questa implementazione, LMCache lavora con vLLM per spostare la cache KV dalla memoria GPU a tier più grandi come la RAM della CPU, il disco locale o lo storage condiviso (come StorageGRID S3 o un mount ONTAP NAS).

Nella configurazione predefinita, vLLM mantiene la cache solo nella memoria GPU. Con l’aumentare del carico di sistema, questo diventa il collo di bottiglia. In questa soluzione, vLLM viene abbinato a LMCache, dove vLLM gestisce il modello, mentre LMCache si occupa del trasferimento e del riutilizzo della cache tra CPU e storage NetApp condiviso. LMCache si connette a vLLM tramite un connettore; lo abiliti all’avvio con il flag --kv -transfer-config di vLLM, così vLLM e LMCache salvano la cache nello storage e la ricaricano in caso di cache hit.

La modalità in-process è il metodo originale per eseguire LMCache con vLLM. Quando usi la modalità in-process, LMCache viene eseguito all’interno del processo vLLM. Le versioni successive di LMCache hanno aggiunto la modalità multiprocess, che è l’approccio attualmente consigliato per un migliore supporto delle funzionalità e prestazioni. Questa sezione tratta la modalità in-process, che è segnata come deprecata nella documentazione attuale di LMCache. Per le nuove implementazioni, valuta invece l’uso della modalità multiprocess.

Per questa implementazione, dovrai:

- Installa vLLM insieme a LMCache
- Avvia vLLM con il connettore LMCache abilitato
- Implementa una configurazione a tre livelli (GPU + CPU + storage condiviso) con due istanze vLLM (su GPU separate) dietro un router vLLM, entrambe che usano lo stesso backend di storage condiviso.

Prerequisiti

- Server Linux con GPU NVIDIA e driver NVIDIA (inclusi CUDA) e almeno una GPU (due GPU o più server per la configurazione completa a due istanze + router)
- Python e uv installati
- Docker e NVIDIA Container Toolkit installati (richiesti per il router vLLM nel passaggio 4)
- Accesso alla rete per scaricare il modello (ad esempio, Hugging Face) e per raggiungere il tuo endpoint di storage

Backend S3 StorageGRID

- Bucket S3 creato per l'uso della cache KV
- Credenziali di lettura/scrittura per il bucket
- Connettività di rete dal server GPU all'endpoint S3
- Richieste S3 virtual hosted-style abilitate

Backend ONTAP pNFS/NFS

- Volume ONTAP esportato sul/sui server GPU
- Percorso di montaggio creato (ad esempio, /mnt/kvcache) su **ogni** server che esegue vLLM. Esempio di comando di montaggio per pNFS dalle performance elevate su 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. Installa vLLM e LMCache

Crea un ambiente virtuale e installa vLLM e LMCache. Consulta la documentazione di LMCache "documentazione di installazione" per ulteriori dettagli. È fondamentale che tu segua il percorso di installazione corretto per la tua versione di CUDA.

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

A questo punto hai il motore di inferenza (vLLM) e il layer di cache (LMCache).

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

vLLM non scarica la cache KV autonomamente. Abiliti LMCache tramite il connettore di trasferimento KV di vLLM. Crea un file YAML (lmcache-config.yaml) che definisce la configurazione della CPU e del tier di storage condiviso. LMCache legge questo file YAML come parte della sequenza di avvio dell'inferenza di vLLM.

Esempio di frammento: StorageGRID S3 shared tier

```
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: sostituisci il nome del bucket, l'endpoint, le credenziali e `disable_tls` per adattarli al tuo ambiente.

Esempio di frammento: tier condiviso ONTAP pNFS/NAS

```
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 l'export ONTAP utilizzando la procedura NFS/pNFS del tuo sito prima del passaggio 3.

[[3-run-two-vllm-instances-shared-cache-across-servers]]
=== 3. Esegui due istanze di vLLM (cache condivisa tra i server)

Imposta le variabili d'ambiente comuni su entrambe le istanze:

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

Istanza 1 (GPU 0, port 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
```

Istanza 2 (GPU 1, port 8002 — terminale separato):

```
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
```

Usa lo stesso file di configurazione e lo stesso hash seed così entrambe le istanze concordano sull'identità del blocco di cache e possono leggere i dati offloadati l'una dall'altra dallo storage condiviso. Imposta `VLLM_API_KEY` su entrambe le istanze; il router passa questo valore come `OPENAI_API_KEY` così le richieste instradate vengono accettate dalle istanze.

Nota: puoi anche utilizzare una singola istanza GPU per implementare una tiered storage architecture. In tal caso salta il passaggio 4.

[[4-deploy-the-vllm-router-single-entry-point]]
=== 4. Distribuisci il router vLLM (punto di ingresso singolo)

Dopo che entrambe le istanze sono in salute, implementa un router così i client usano un URL compatibile 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
```

Invia traffico a http://localhost:8000/v1. Il router distribuisce le richieste; LMCache e lo storage condiviso consentono il riutilizzo della cache tra le istanze, non solo all'interno di una singola GPU.

Ad esempio, puoi inviare una richiesta al router utilizzando curl nel modo seguente (sostituisci <shared_api_key> con la tua rispettiva chiave 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. Verifica che il tiering funzioni

- Entrambi i processi vLLM sono in ascolto su 8001 e 8002; il router risponde su 8000.
- Invia una richiesta con un prefisso lungo attraverso il router.
- Conferma che gli oggetti o i file compaiano nello storage condiviso (bucket S3 o ls -ltr /mnt/kvcache).
- Invia nuovamente una richiesta simile: la seconda richiesta dovrebbe mostrare un precaricamento più rapido o un comportamento di cache hit nei log.
- Ripeti attraverso il router così il traffico può raggiungere l'altra istanza.

Conclusione

L'implementazione di una tiered KV cache architecture sposta la KV cache dalla memoria GPU alla memoria CPU e allo storage condiviso NetApp, così l'inferenza può scalare con gli utenti e la lunghezza del contesto senza sprecare cicli GPU in ripetuti prefill. Lo storage condiviso trasforma la cache in un'infrastruttura riutilizzabile tra le istanze di serving e ti aiuta a ottenere più valore dal tuo investimento GPU esistente.

NetApp storage è perfetto per questo tier perché offre tutto ciò di cui l'inferenza di produzione ha bisogno oltre alla capacità raw: accesso standard tramite S3 e NFS/pNFS, cache condivisa che più istanze del motore di servizio possono usare contemporaneamente e servizi dati enterprise su cui già fai affidamento per durabilità, protezione e governance. Non ti serve un client proprietario; i framework di offload della cache KV si connettono a StorageGRID S3 o a un mount NAS ONTAP usando protocolli familiari.

NetApp ti offre una base storage familiare per l'offload della cache KV senza cambiare il modo in cui esegui l'inferenza o il modo in cui i tuoi team gestiscono i dati.

Informazioni sul copyright

Informazioni sul marchio commerciale