KV Cache 卸载与 NetApp

NetApp artificial intelligence solutions

NetApp
August 20, 2026

目录

KV Cache 卸载与 NetApp

第一波企业人工智能投资侧重于模型训练和微调构建能力，而不是大规模服务。随着组织从实验转向生产，重心转向实时推理：用户不断互动的搜索助手、聊天机器人、copilot 和代理工作流程。在这些环境中，GPU 利用率快速上升，并不是因为每个请求都运行完整的训练过程，而是因为每个后续问题、对话中的每个新轮次以及每个并发用户都会为推理堆栈增加负载。

简介

一种模式很快就会显现出来：GPU 正在做大量令人惊讶的重复工作，重新计算它已经处理过的令牌的注意力。这种重复不仅仅是一个调优问题；它是一个内存架构问题。业界的回应是围绕 KV 缓存构建的分层内存策略。

什么是 **KV Cache**?

简而言之，KV（键值）缓存是一种通过将先前计算的值存储在 KV 缓存中来优化大型语言模型（LLM）推理的技术，这样就无需为每个新生成的 token 重新计算这些值，否则每次都需要重新计算。

注：有关更深入的技术概述，请查看 "Hugging Face KV 缓存概述"。

什么是 **KV Cache Offloading**?

默认情况下，大多数推理引擎将 KV 缓存存储在 GPU 内存中。这在需求增长之前效果很好。KV 缓存大小随批量大小（同时处理的提示数）和序列长度（每个会话或生成流的长度）线性扩展。随着上下文窗口的扩展，多轮聊天变得普遍，更多的用户和代理使用推理服务，KV 缓存需求可能会迅速超出可用的 GPU 内存。发生这种情况时，有用的缓存条目通常会被逐出，引擎必须重新计算已处理令牌的注意力。

KV 缓存卸载通过将先前处理的提示的 KV 缓存移动到 GPU 或加速器内存（如系统 RAM、本地磁盘或共享存储）之外的外部目标来解决这一约束。只要容量允许，卸载的条目可以保留，并在类似的前缀或后续请求到达时再次引用，而不是在 GPU 内存填满时被丢弃。实际上，这些卸载目标充当 GPU 内存的分层交换空间：比 VRAM 慢，但容量更大、更持久，从而扩展了系统在无需重复预填充工作的情况下可以维持的会话上下文量和并发工作负载。

共享存储层的重要性

当单个推理引擎超出 GPU 内存时，KV 缓存卸载已能发挥作用。将缓存块移至 CPU RAM 可扩展单台服务器的容量。这虽然有用，但仅解决了部分生产问题。真正的推理平台很少以单个服务引擎实例的形式运行。它们运行于负载均衡器之后，水平扩展，并为大量并发用户和代理提供服务。在这种环境中，共享存储正是将 KV 缓存从本地优化提升为平台能力的关键所在。

- 共享存储支持跨服务实例重用 共享存储的主要优势之一是能够跨多个实例和节点访问相同的文件或对象。在负载均衡器后面扩展部署两个或更多服务引擎实例的情况下尤其如此，每个实例都被配置为将 KV 缓存块卸载到相同的共享存储桶或 NAS 卷。例如，当一个实例处理提示前缀并卸载生成的缓存块时，另一个实例可以在缓存命中时加载这些块，而不是重新计算相同的预填充工作。这一点很重要，因为生产流量是分布式的，用户的第一个问题可能会命中实例 A；后续问题或具有类似系统提示的另一个用户可能会命中实例 B。如果没有共享存储，每个实例都会维护自己的私有缓存空间。
- 仅有 **CPU** 内存不足以进行多实例部署 CPU 层速度很快，但它对每个服务引擎进程都是本地的。一个实例无法访问另一个实例卸载到其本地 CPU RAM 的 KV 缓存条目，除非您实施对等通道，这会引入额外的延迟和

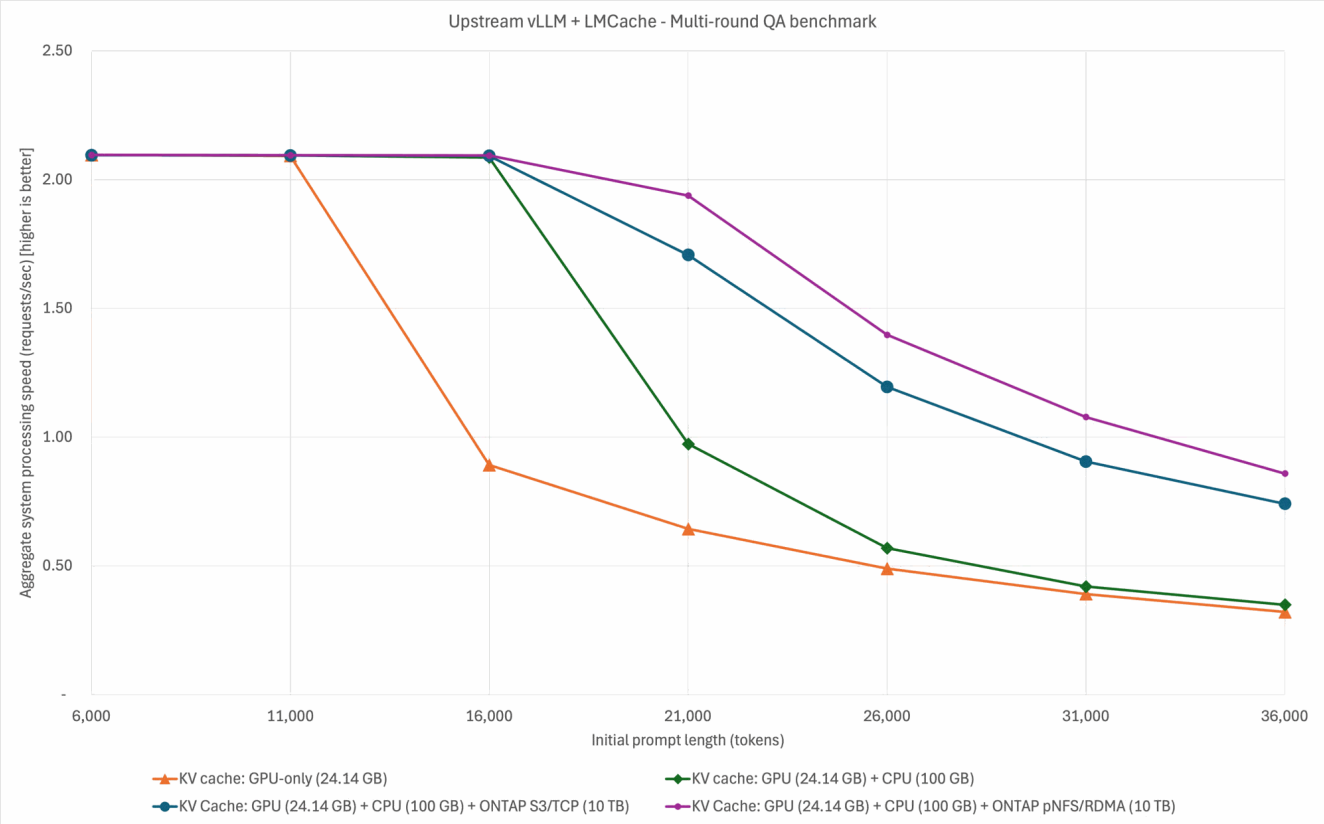
操作复杂性。共享存储消除了这一障碍。CPU RAM 作为每个节点上的快速暂存层仍然很有价值，但共享 S3 或 NAS 提供了多个引擎可以读写的通用内存平面。您不再孤立地扩展 GPU，而是使用共享缓存基础架构进行扩展。

- 实现三层设计策略 + 首选架构结合：
  - **GPU memory** — 用于运行中请求的活动缓存。
  - **CPU**内存 — 随着提示进入队列而快速暂存。
  - 共用存储空间 — 持久、大容量、可共享的后备存储。

  通过这种架构，当新请求到达时，KV 缓存块可以从共享层暂存到 CPU 内存中，从而使 GPU 专注于活动工作，同时仍在存储上保留持久缓存。

- 推理的经济性，而不仅仅是速度 从生产的角度来看，共享存储的重要性不仅仅是延迟。它是对内存、并发性和成本的控制。像 vLLM 这样的服务引擎在单个节点内高效地管理 KV 缓存。像 LMCache 这样的 KV 缓存卸载框架将 KV 缓存转变为可跨 CPU 内存和存储移动的便携式可共享资产。像 NetApp ONTAP 和 StorageGRID 这样的共享存储平台提供了持久层，使跨多个实例的共享切实可行。通过将 LMCache 连接到共享存储，多个 vLLM 实例可以访问相同的已卸载 KV 缓存条目，从而随着并发量的增长提高吞吐量并减少冗余计算。这直接减少了重复预填充所浪费的 GPU 周期，与聊天机器人、搜索助手、代理以及任何具有多轮对话线程、共享系统提示或重复上下文模式的工作负载密切相关。
- 提高推理性能和 **GPU** 投资 + 此基准测试比较了随会话长度和并发负载增加时的推理性能。当 KV 缓存仅保留在 GPU 内存中时，可用余量会迅速耗尽，吞吐量下降（橙色线）。借助三层设计（GPU 用于活动工作、CPU 用于快速暂存、共享存储用于持久且可共享的缓存），该平台可维持更多会话，并避免同样急剧的性能下降。在实践中，分层通过减少重复的预填充计算，帮助您从相同的 GPU 中获得更多工作负载。
  - 简而言之：
    - **GPU tier** — 处理活动的、动态进行中的工作。
    - **CPU tier** — 使最近使用的缓存靠近服务引擎。
    - 共享存储层 — 跨服务器保留缓存，并为新会话释放 GPU/CPU 内存。

  图 1 - 多轮推理基准



该基准表明，在 CPU 内存旁边添加共享存储不会降低性能；它在吞吐量下降出现之前扩展了可用的工作范围。分层有效地为推理添加了内存层，从而减少了每次会话计数、上下文长度或并发量增加时添加 GPU 的需求。

- 企业版：标准协议，无专有客户端 + 共享存储很重要，但并非所有共享存储都是相同的。NetApp 支持使用行业标准协议和工具进行 KV 缓存卸载：
  - **StorageGRID S3**
  - 适用于 **ONTAP NAS** 的 **NFS/pNFS**

  无需自定义专有推理客户端。运营团队可以使用与其他数据服务相同的存储协议，以及熟悉的数据保护、安全性和多云移动性。

部署 KV 缓存卸载

共享存储扩展了 KV 缓存容量，并实现了跨服务实例的重用。要在生产中使用该层，您需要配置推理引擎和 KV 缓存卸载框架。以下各节介绍如何使用常见工具选项实现此堆栈。

vLLM

vLLM 是一种流行的高性能开源 LLM 推理引擎。在此解决方案中，它是在推理期间接收用户请求、生成响应和管理 GPU 内存中 KV 缓存的引擎。vLLM 是可插拔的——您可以选择要使用的卸载框架。以下部分介绍如何使用流行的卸载框架实现基于 vLLM 的服务堆栈。

LMCache（进程内模式）

LMCache 是一种流行的开源 KV 缓存卸载框架。本节介绍如何实现使用 LMCache 进行 KV 缓存卸载的基于 vLLM 的服务堆栈。在此实现中，LMCache 与 vLLM 协同工作，将 KV 缓存从 GPU 内存移动到更大的层级，例如 CPU RAM、本地磁盘或共享存储（如 StorageGRID S3 或 ONTAP NAS 挂载）。

在默认设置中，vLLM 仅在 GPU 内存中保留 KV 缓存。随着系统负载的增加，这将成为瓶颈。在此解决方案中，vLLM 与 LMCache 配对，其中 vLLM 提供模型，而 LMCache 跨 CPU 和共享 NetApp 存储卸载和重用缓存。LMCache 通过连接器连接到 vLLM；启动时使用 vLLM 的 --kv-transfer-config 标志启用它，从而使 vLLM 和 LMCache 将缓存保存到存储中，并在缓存命中时重新加载。

进程内模式是使用 vLLM 运行 LMCache 的原始方法。使用进程内模式时，LMCache 在 vLLM 进程中运行。后来的 LMCache 版本添加了多进程模式，这是目前推荐的方法，可获得更好的功能支持和性能。本节介绍进程内模式，该模式在当前 LMCache 文档中标记为已弃用。对于新部署，请考虑改用多进程模式。

对于此部署，您将：

- 与 LMCache 一起安装 vLLM
- 启用 LMCache 连接器后启动 vLLM
- 在 vLLM 路由器后面部署具有两个 vLLM 实例（在单独的 GPU 上）的三层设置（GPU + CPU + 共享存储），两者均使用相同的共享存储后端。

前提条件

- 具有 NVIDIA 驱动程序（包括 CUDA）和至少一个 GPU 的 Linux GPU 服务器（两个 GPU 或多个服务器用于完整的双实例 + 路由器布局）
- 已安装 Python 和 uv
- 已安装 Docker 和 NVIDIA Container Toolkit（步骤 4 中的 vLLM 路由器需要）
- 通过网络访问下载模型（例如 Hugging Face）并访问您的存储端点

StorageGRID S3后端

- 为 KV 缓存使用创建的 S3 存储桶
- 存储桶的读/写凭据
- 从 GPU 服务器到 S3 端点的网络连接
- 虚拟托管样式 S3 请求已启用

ONTAP pNFS/NFS 后端

- 导出到 GPU 服务器的 ONTAP 卷
- 在运行 vLLM 的*每个*服务器上创建的挂载路径（例如 /mnt/kvcache）。适用于基于 RDMA (RoCE) 的高性能 pNFS 的示例挂载命令：

```
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.安装 vLLM 和 LMCache

创建虚拟环境并安装 vLLM 和 LMCache。有关更多详细信息，请参阅 LMCache "安装文档"。请务必遵循适合您的 CUDA 版本的正确安装路径。

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

此时，您有推理引擎（vLLM）和缓存层（LMCache）。

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

vLLM 不会自行卸载 KV 缓存。您可以通过 vLLM 的 KV 传输连接器启用 LMCache。创建定义 CPU 和共享存储层配置的 YAML 文件 (lmcache-config.yaml)。LMCache 读取此 YAML 文件作为 vLLM 推理启动序列的一部分。

示例代码段：**StorageGRID S3** 共享层

```
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"
```

注：替换存储桶名称、端点、凭据和 `disable_tls`以匹配您的环境。

示例代码段：**ONTAP pNFS/NAS** 共享层

```
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
```

在步骤 3 之前，使用您站点的 NFS/pNFS 程序挂载 ONTAP 导出。

[[3-run-two-vllm-instances-shared-cache-across-servers]]
=== 3.运行两个 vLLM 实例（跨服务器共享缓存）

在两个实例上设置通用环境变量：

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

Instance 1（GPU 0，端口 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
```

Instance 2（GPU 1，端口 8002 — 独立终端）：

```
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
```

使用相同的配置文件和哈希种子，以便两个实例在缓存块标识上达成一致，并可以从共享存储中读取彼此卸载的数据。在两个实例上设置 `VLLM_API_KEY`；路由器将此值作为 `OPENAI_API_KEY`传递，以便路由的请求被实例接受。

注：单个 GPU 实例也可用于实现分层存储架构。在这种情况下，跳过步骤 4。

[[4-deploy-the-vllm-router-single-entry-point]]
=== 4.部署 vLLM 路由器（单入口点）

在两个实例均正常运行后，部署路由器，以便客户端使用一个与 OpenAI 兼容的 URL。

```
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
```

将流量发送到 http://localhost:8000/v1。路由器分发请求；LMCache 和共享存储允许跨实例重用缓存，而不仅仅是在一个 GPU 内。

例如，您可以使用以下 curl 向路由器发送请求（将 <shared_api_key> 替换为相应的 API 密钥）：

```
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.验证分层是否正常工作

- 两个 vLLM 进程分别侦听 8001 和 8002；路由器在 8000 上响应。
- 通过路由器发送带有长前缀的请求。
- 确认对象或文件出现在共享存储（S3 存储桶或 ls -ltr /mnt/kvcache）上。
- 再次发送类似的请求——第二个请求应在日志中显示更快的预填充或缓存命中行为。
- 通过路由器重复此操作，以便流量可以到达另一个实例。

结束语

部署分层 KV 缓存架构将 KV 缓存从 GPU 内存移动到 CPU 内存和共享 NetApp 存储，因此推理可以随用户和上下文长度扩展，而无需在重复预填充时浪费 GPU 周期。共享存储将缓存转化为跨服务实例的可重用基础架构，并帮助您从现有 GPU 投资中获得更多价值。

NetApp 存储非常适合这一层，因为它提供了超出原始容量的生产推理所需的功能：通过 S3 和 NFS/pNFS 进行标准访问、多个服务引擎实例可同时使用的共享缓存，以及您已依赖的用于持久性、保护和治理的企业数据服务。您无需专有客户端；KV 缓存卸载框架使用熟悉的协议连接到 StorageGRID S3 或 ONTAP NAS 挂载点。

NetApp 为 KV 缓存卸载提供了熟悉的存储基础，而无需更改运行推理的方式或团队管理数据的方式。