

KV Cache (DRAM 与 CMX) 分层落点对比 测试报告

绿算 GP7000 • DeepSeek-V4-Flash • vLLM(dynamo) 聚合推理集群

广东省绿算技术有限公司

2026 年 10 月 13 日

目 录

1. 测试结构	4
1.1 结构图	4
1.2 分层说明	5
1.3 vLLM 与 Mooncake 的连接	7
1.4 网络通路矩阵	9
1.5 端口与地址总表	9
1.6 两种配置的结构差异	10
2. 测试环境	10
2.1 物理机	10
2.2 GPU 拓扑	11
2.3 网络	11
2.4 本地存储	12
2.5 控制机与模型	12
3. 配置	12
3.1 容量基线说明	12
3.2 引擎启动参数	12
3.3 引擎环境变量	13
3.4 store 配置	14
3.5 master 配置	15
3.6 负载形状 (共 5 种)	15
3.7 测量口径	16
3.8 两组负载的健康判据	16
3.9 切模式与核验命令	16
4. 性能对比	17
4.1 纯读路径	17
4.2 对比的有效性	17
4.3 20 轮 agent 会话 (osl=128, C=4...128)	18
4.4 更高并发 (C=256, 20 轮)	19
4.5 每轮墙钟的加法	19
4.6 批次形状效应 (两模式同时出现, 与介质无关)	20
5. 多轮对话	20

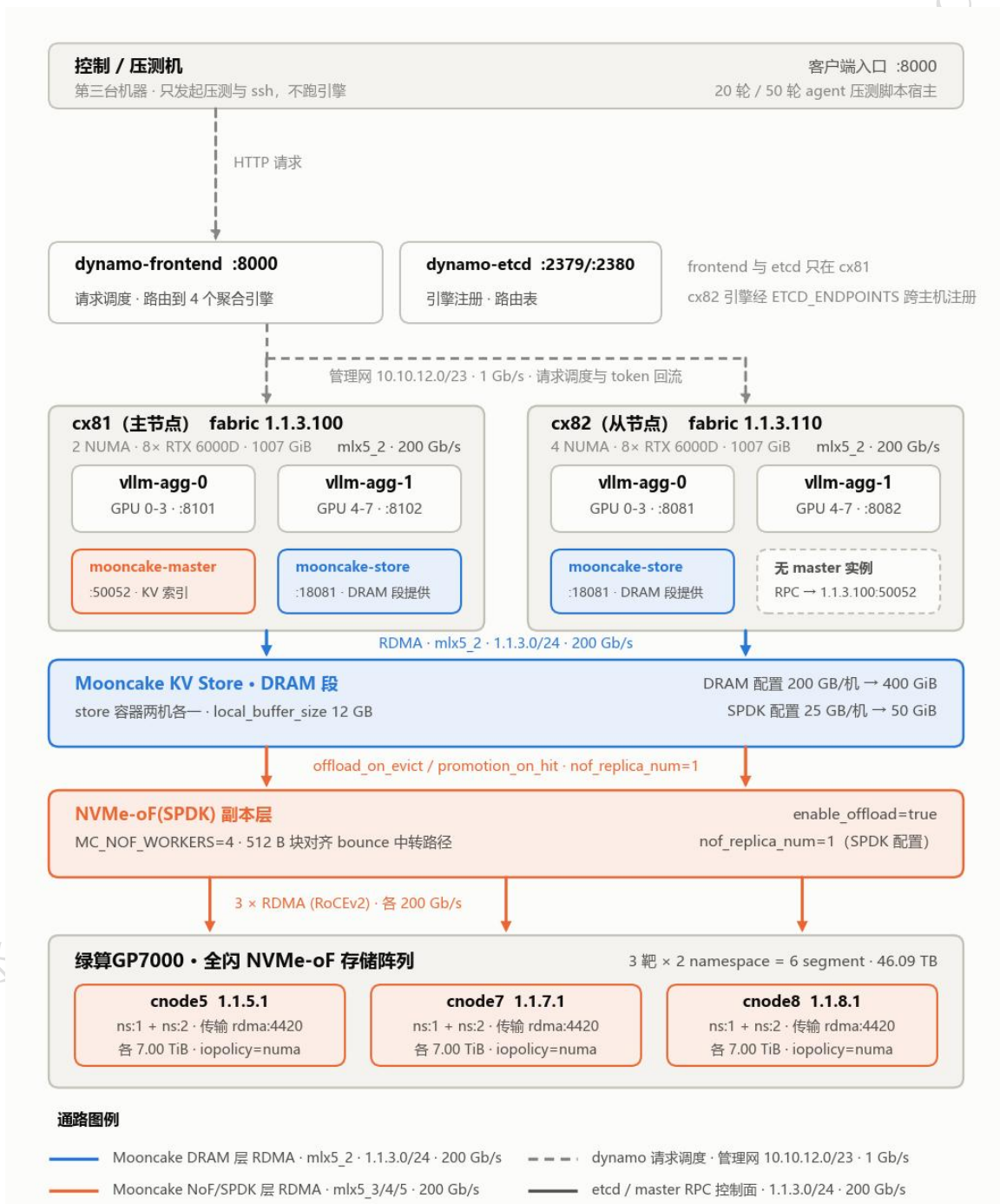
5.1 容量: $107\times$	20
5.2 20 轮	20
5.3 工作集超出 DRAM 容量	21
5.4 长输出 (osl=5000) : $112\times$	21
5.5 50 轮超长会话	23
5.6 收益的来源	23
6. 结论	24
收益	25

1. 测试结构

本章给出被测系统的完整拓扑：从发起压测的客户端入口，经 dynamo 调度层与 4 个聚合 vLLM 引擎，落到 Mooncake KV Store 的 DRAM 段与 NVMe-oF (SPDK) 副本层，最终落到 绿算 GP7000 全闪 NVMe-oF 存储阵列。

被测拓扑由三台机器构成：测试服务器（控制/压测机，只发起请求）、cx81（主节点，兼 master + frontend + etcd）、cx82（从节点）。两节点各 8 张 RTX 6000D，各跑 2 个聚合引擎，共 4 个 vLLM 引擎；KV 面统一挂在一个 200 Gb/s 的 RoCEv2 fabric 上。

1.1 结构图



测试结构：dynamo 调度 + vLLM 聚合引擎 + NVMe-oF (绿算 GP7000)

图中颜色语义：

颜色	含义
蓝 #2a78d6	Mooncake DRAM 层——DRAM 段及其 RDMA 通路
橙 #eb6834	Mooncake NoF/SPDK 层——NVMe-oF 副本层、master、绿算 GP7000
•	控制面 RPC (etcd / master)
浅灰虚线	dynamo 请求调度面 (1 Gb/s 管理网)

1.2 分层说明

1.2.1 客户端与调度层 (dynamo)

组件	位置	监听	作用
控制 / 压测机 spark-170	第三台机器	—	只发起压测与 ssh, 不跑引擎; 20 轮 / 50 轮 agent 压测脚本宿主
dynamo-frontend	cx81	:8000	客户端唯一入口; 请求调度、路由到 4 个聚合引擎、token 回流
dynamo-etcd	cx81	:2379 / :2380	引擎注册与路由表

frontend 与 etcd 只在 cx81; cx82 的两个引擎通过 ETCD_ENDPOINTS 跨主机注册。

调度面走管理网 10.10.12.0/23, 1 Gb/s —— 是整条拓扑里唯一不是 200 Gb/s 的一条腿, 与 KV 面相差 200×。

1.2.2 引擎层 (vLLM 聚合引擎)

主机	容器	镜像	物理 GPU	端口
cx81	vllm-agg-0	ds4-vllm:nof-aff-20260930	0-3	:8101
cx81	vllm-agg-1	ds4-vllm:nof-aff-20260930	4-7	:8102
cx82	vllm-agg-0	ds4-vllm:nof-aff-20260930	0-3	:8081
cx82	vllm-agg-1	ds4-vllm:nof-aff-20260930	4-7	:8082

4 个引擎的启动参数完全一致, 唯一差异是 DYN_SYSTEM_PORT (8101 / 8102 / 8081 / 8082)。

每个引擎独占本机 4 张物理 GPU, 聚合式部署: 一台 8 卡机器拆成 2 个 4 卡实例。

每卡: NVIDIA RTX 6000D (Blackwell, sm_120), 卡间无 NVLink (8 卡全部走 PCIe); 完整硬件参数与 P2P 实测带宽见 § 2.2 GPU 拓扑。

每机 8 张卡全部注册 GPUDirect (nvidia_peermem refcount = 8), KV 直接以 GPU 显存地址进出 RDMA。

模型在两机各自本地 (非共享存储), 每机 2 个引擎直接读本机磁盘, 不存在共享存储依赖; 路径、分片数与 JIT 缓存见 § 2.5 控制机与模型。

1.2.3 KV 存储层 (Mooncake KV Store)

组件	位置	端口	说明
mooncake-master-50052	cx81 (唯一实例)	:50052 RPC • :19101 metrics	KV 索引唯一实例；cx82 侧 RPC → 1.1.3.100:50052
mooncake-store-50052	cx81 / cx82 各一	:18081	提供本机 DRAM 段；两机镜像不同（各与本机引擎镜像一致）

DRAM 段容量（两种配置的结构性差异所在）：

	DRAM 配置	SPDK 配置
global_segment_size	200GB/机	25GB/机
实际段大小（二进制解析）	400 GiB	50 GiB

"GB" 被 Mooncake 按二进制解析，且该值是每机的 $\Rightarrow$ 段总量 = $2 \times$ 标称值。部署后必须用 master_total_capacity_bytes 核验。

引擎侧 KV 配置 (vllm.json / kv-aggr.json)：

```
{ "metadata_server": "P2PHANDSHAKE",
  "master_server_address": "1.1.3.100:50052",
  "protocol": "rdma",
  "device_name": "mlx5_2",
  "mode": "standalone-store",
  "global_segment_size": 0,
  "local_buffer_size": "1GB",
  "enable_offload": true }
```

global_segment_size: 0 $\Rightarrow$ 引擎不提供 DRAM 段，只做客户端；段全部由 store 进程提供。

enable_offload: true 是 SPDK 初始化的入口——为真即走 SPDK 路径。

1.2.4 存储阵列层 (NVMe-oF • 绿算 GP7000)

项	值
设备	绿算 GP7000 (全闪 NVMe-oF 存储阵列)
靶 (cnode)	cnode5 1.1.5.1 / cnode7 1.1.7.1 / cnode8 1.1.8.1
传输	rdma:4420 (RoCEv2)
namespace	每靶 2 个 (ns:1 / ns:2), iopolicy=numa
单 ns 容量	$15,002,931,888 \times 512 \text{ B} = 7,681,501,126,656 \text{ B}$ (7.00 TiB)
NoF 池总量	$3 \text{ 靶} \times 2 \text{ ns} = 6 \text{ segment} = 46,089,006,759,936 \text{ B} = 42.0 \text{ TiB} = 46.09 \text{ TB}$ (master_total_nof_capacity_bytes)

RDMA 链路 靶与主机一一对标: cnode5 $\leftrightarrow$ mlx5_3 / cnode7 $\leftrightarrow$ mlx5_4 / cnode8 $\leftrightarrow$ mlx5_5, 与 DRAM 层的 mlx5_2 物理分离

启动器参数 NOF=1 NOF_REPLICAS=1 NOF_NSIDS=1,2 NOF_NS_SIZE=7681501126656

NVMe-oF 会话 (nvme list-subsys, 两机均建立)：

subsys	NQN	traddr	传输	状态	入 NoF 池
nvme5	···:cnode5	1.1.5.1	rdma:4420	live	☒
nvme6	···:cnode7	1.1.7.1	rdma:4420	live	☒
nvme7	···:cnode8	1.1.8.1	rdma:4420	live	☒

subsys	NQN	traddr	传输	状态	入 NoF 池
nvme8	···:cnode9	1.1.9.1	rdma:4420	live	✗
nvme9	···:cnode10	1.1.10.1	rdma:4420	live	✗

SPDK 路径的块对齐约束（本层的结构性工程约束）：

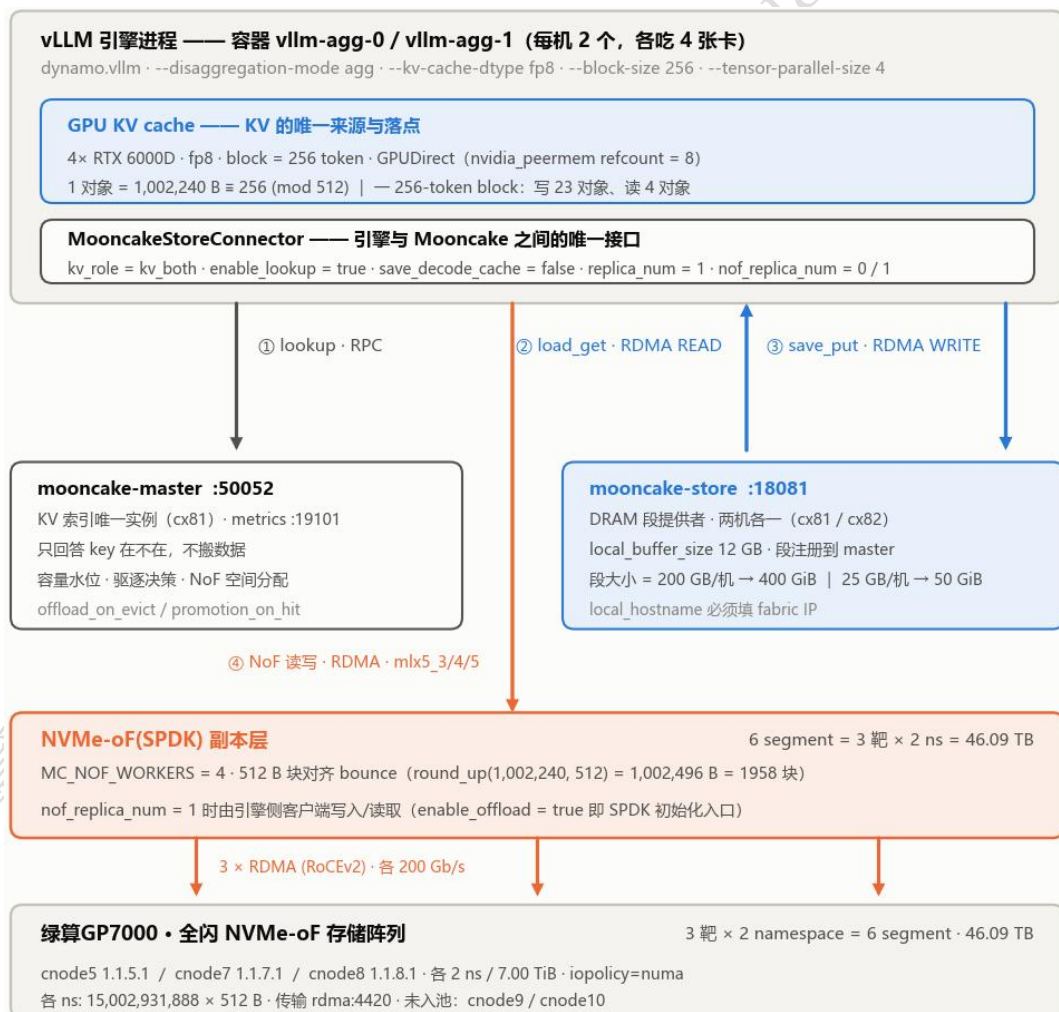
NoF 数据面硬校验 offset / size / ptr 三者全部 512 B 对齐（transfer_task.cpp:1435-1448，NOF_BLOCK_ALIGNMENT = 512，无配置项可绕过）。

而 DSV4 的 KV 对象尺寸恒为 1,002,240 B $\equiv$ 256 (mod 512) $\Rightarrow$ 几何上必然失配。

因此在 Mooncake 数据面加了一条块对齐 bounce buffer 路径：未对齐时经 SPDK 注册的对齐 scratch 中转（长度 round_up(size, 512) = 1958 块），读路径再拷回原始字节数。

1.3 vLLM 与 Mooncake 的连接

vllm 推理引擎于 Mooncake 连接，vllm 上层只有一个挂载点（connector），挂载点之下是两条并行的进程间通路（控制面 $\rightarrow$ master，数据面 $\rightarrow$ store），第三条是引擎侧直接对 NVMe-oF 副本层发起的 NoF 通路。



vLLM 引擎与 Mooncake 的连接：一个 connector、四类调用、两条 RDMA 通路

1.3.1 连接点

引擎侧与 Mooncake 的全部联系都收在一个启动参数上：

```
{ "kv_connector": "MooncakeStoreConnector", "kv_role": "kv_both",
  "kv_connector_extra_config": { "enable_lookup": true, "save_decode_cache": false,
  "mooncake_protocol": "rdma", "replica_num": 1, "nof_replica_num": 0 }}
```

kv_role: kv_both ⇒ 引擎既读也写；enable_lookup: true ⇒ prefill 之前先查一次 KV 索引。

不挂这条参数，引擎根本不连 Mooncake

引擎是纯客户端：引擎侧 vllm.json 里 mode: "standalone-store" + global_segment_size: 0 ⇒ 引擎不提供 DRAM 段，段全部由两机的 store 进程提供。引擎这一侧只有 GPU 显存里的 KV，以及 local_buffer_size: "1GB" 的注册中转缓冲。

调用	发起方 → 对端	载体	功能
lookup	引擎 → master	控制面 RPC • 1.1.3.100:50052	查索引：本次请求的前缀里哪些 block 已有 KV
load_get	引擎 ← DRAM 段 (store)	DRAM 层 RDMA • mlx5_2	把命中的 KV 读进 GPU 显存；DRAM 段已驱逐时改由 NoF 副本层供给
save_put	引擎 → DRAM 段 (store)	DRAM 层 RDMA • mlx5_2	prefill 算完，把 KV 落成 DRAM 段里的主副本 (replica_num: 1)
NoF 读写	引擎 → NoF 副本层	NoF 层 RDMA • mlx5_3/4/5	nof_replica_num: 1 时同一份 KV 再写一份副本、命中时从副本读

1.3.2 控制面与数据面是两条独立通路

	控制面 (①)	数据面 (②③)
对端	mooncake-master	mooncake-store
端口	:50052 RPC (:19101 只出 metrics)	:18081
传什么	KV 索引：key 在不在	KV 正文：在引擎与 store 的已注册内存之间直搬
网段 / 网卡	1.1.3.0/24 • mlx5_2	1.1.3.0/24 • mlx5_2

正文从不经过 master: master 只是索引与决策者 (--enable_offload=false)，因此它的负载与写入字节数无关。

对等握手：metadata_server: "P2PHANDSHAKE" ⇒ 客户端与段提供者之间对等握手，不走 etcd 那套元数据服务。正因如此 local_hostname 必须填 fabric 字面 IP；填管理 IP 时握手照样成功、搬数据时静默返回空，tier 恒 0 B 且不报错。

GPUDirect: nvidia_peermem refcount = 8 ⇒ 每机 8 张卡的显存全部注册进 RDMA，KV 以 GPU 地址直接进出网卡，不经主机内存中转。这也解释了为什么 bounce 拷贝必须走 device::RuntimeAccelerator —— 源指针在显存里。

1.3.3 一个配置项落在哪个进程

配置	谁读它	关键字段	改完必须重启
连接器与角色	引擎 argv	kv_connector / kv_role / replica_num / nof_replica_num / save_decode_cache	引擎
引擎侧 Mooncake	引擎 env → vllm.json	master_server_address / device_name / mode / enable_offload	引擎
段大小与客户端身份	store → store.json	global_segment_size / local_hostname / local_buffer_size	store
索引与策略	master 命令行	--rpc_port / -- offload_on_evict / -- promotion_on_hit / -- default_kv_lease_ttl	master (生产索引在内存中, 本次测试全程未重启)

四者互不覆盖：只改 store.json 的段大小而不重启 store，引擎仍连在旧段上；只改 nof_replica_num 而不重启引擎，argv 不变、配置不生效。

1.4 网络通路矩阵

#	通路	网段	适配器	速率
1	dynamo 请求调度 (frontend ↔ worker, 含 token 回 流)	10.10.12.0/23 (管理 网)	板载	1 Gb/s
2	etcd / master RPC (控制面)	1.1.3.0/24	mlx5_2	200 Gb/s
3	Mooncake DRAM 层 RDMA	1.1.3.0/24	mlx5_2	200 Gb/s
4	Mooncake NoF/SPDK 层 RDMA	1.1.5 / 1.1.7 / 1.1.8 各一条	mlx5_3 / mlx5_4 / mlx5_5	各 200 Gb/s

链路层全部是 Ethernet (RoCEv2)。

两个 tier 的 RDMA 走不同的物理网卡 ⇒ 介质 A/B 对比时 DRAM 层与 NoF 层不互抢带宽。

store.json / vllm.json 中 device_name: "mlx5_2" ⇒ Mooncake 的 DRAM 层与控制面共用一个 200 Gb/s 平面。

1.5 端口与地址总表

主机	container / 服务	端口	fabric IP	镜像
cx81	dynamo-frontend	:8000 (客户端入口)	1.1.3.100	nof-bouncepool2- 20260927
cx81	dynamo-etcd	:2379 / :2380	1.1.3.100	nof-bouncepool2- 20260927
cx81	mooncake-master- 50052	:50052 RPC / :19101 metrics	1.1.3.100	nof-align-20260927
cx81	mooncake-store-50052	:18081	1.1.3.100	nof-align-20260927

主机	container / 服务	端口	fabric IP	镜像
cx81	vllm-agg-0 / vllm-agg-1	:8101 / :8102	1.1.3.100	ds4-vllm:nof-aff-20260930
cx82	mooncake-store-50052	:18081	1.1.3.110	nof-bouncepool2-20260927
cx82	vllm-agg-0 / vllm-agg-1	:8081 / :8082	1.1.3.110	ds4-vllm:nof-aff-20260930
spark-170	控制 / 压测机	—	—	—

两机 store 镜像不同，各自与本机引擎镜像一致。

同一 tag 在两机对应不同 Image ID (726bf0788423 / 41f8f1630e91) —— 各自本机构建、patchelf 改哈希 ⇒ 跨机不要假设字节一致。

主机	管理 IP	带外 IP
cx81	100.105.170.73	10.10.13.27/23
cx82	100.108.48.48	10.10.12.107/23

local_hostname 必须填 fabric 字面 IP (1.1.3.x)；写成管理 IP 会让每次 Put 静默失败、tier 恒为 0 B，不报错。

1.6 两种配置的结构差异

两种配置共用同一套拓扑，唯一的引擎侧变量是 KV 副本层是否启用，唯一的 store 侧变量是 DRAM 段大小：

结构项	DRAM 配置	SPDK 配置
replica_num	1	1
nof_replica_num	0	1
DRAM 段 / 机	200 GB → 400 GiB	25 GB → 50 GiB
NoF 副本层	不启用（绿算 GP7000 空闲）	启用（6 segment / 46.09 TB）
存储侧拓扑	纯 DRAM 段	DRAM 段 + 绿算 GP7000

replica_num / nof_replica_num / --max-num-seqs 三者都住在引擎 argv 里 ⇒ 任何一项变更必须重启引擎才生效；只改 store.json 对已运行的引擎无效。

2. 测试环境

2.1 物理机

● 算力服务器

项	cx81 (主节点)	cx82 (从节点)
角色	master + store + 2 引擎 + frontend + etcd	store + 2 引擎
CPU	Intel Xeon 6767P, 2 socket × 64 core × 2HT = 256 vCPU	Intel Xeon 6787P, 2 × 86 × 2HT = 344 vCPU
NUMA	2 nodes (515,429 / 516,039 MB)	4 nodes (各 ≈257.4 - 258.0 GB)
内存	1007 GiB	1007 GiB

项	cx81 (主节点)	cx82 (从节点)
大页	24576 × 2 MiB = 48 GiB	49152 × 2 MiB = 96 GiB
OS / 内核	Ubuntu 24.04.4 LTS / 6.8.0-137-generic	同
/raid	/dev/md0 ext4 stripe=512	同

各机 fabric / 管理 / 带外地址见 § 1.5 端口与地址总表。

● G3.5 存储 GP-7000

项	值
主控器	PCIe 5.0 主控*2
业务板卡数量	NVIDIA BlueField 3*6
端口数量	200Gbps*12 Roce V2
SSD 数量	至誉 PD5 pro PCIe 5.0 U.2 SSD*24
SSD 容量	7.68TB*24
Nvme Target	SPDK

2.2 GPU 拓扑

项	值
型号	NVIDIA RTX 6000D (Blackwell, sm_120)
数量 / 显存	每台 8 张 / 85,651 MiB (83.6 GiB) 每张
SM 数	156
显存带宽 (实测持续)	1278.7 GB/s (GDDR7 448-bit)
驱动 / 功耗上限	595.91.07 (两机一致) / 600.00 W
PCIe	Gen5 x16
NVLink	无 —— 8 卡之间全部走 PCIe (nvidia-smi topo -m 无任何 NV#)
NUMA 归属	GPU0-3 → node0; GPU4-7 → node1(cx81) / node2(cx82)

卡间通信 (无 NVLink):

指标	值
P2P 可达性	8×8 全通 (除对角线)
点对点单向带宽	~36.7 GB/s (PXB/NODE/SYS 三种拓扑几乎无差)
同桥卡对 (0-1/2-3/4-5/6-7) 双向合计	仅 37.7 GB/s (不能全双工)
跨桥卡对双向	49.3 - 65.8 GB/s
NCCL all_reduce (world=8, 256 MB)	11.2 GB/s busbw
2 卡 → 8 卡的广播	30.5 → 9.8 → 10.1 GB/s (每卡可用带宽随规模急剧下降)
经主机中转 vs 直连	直连快 1.28 - 1.30× (确证走真 peer DMA)

2.3 网络

HCA	netdev	速率	状态	cx81 IP	角色
mlx5_0 / mlx5_1	ens1f*	100 Gb/s	PORT_DOWN	—	未启用

HCA	netdev	速率	状态	cx81 IP	角色
mlx5_2	enp61s0f0np0	200 Gb/s	ACTIVE	1.1.3.100	主 RDMA 平面 (DRAM 层 + 控制面)
mlx5_3	enp61s0f1np1	200 Gb/s	ACTIVE	1.1.5.100	与 GP 7000 1.1.5.1 同段
mlx5_4	enp78s0f0np0	200 Gb/s	ACTIVE	1.1.7.100	与 GP 7000 1.1.7.1 同段
mlx5_5	enp78s0f1np1	200 Gb/s	ACTIVE	1.1.8.100	与 GP 7000 1.1.8.1 同段
mlx5_6 / mlx5_7	enp186*	200 Gb/s	ACTIVE	1.1.9 / 1.1.10	连了,但未入 GP 7000 池
mlx5_8 / mlx5_9	enp203*	200 Gb/s	ACTIVE	1.1.11 / 1.1.12	备用

四条通路 (dynamo 调度面 / etcd • master 控制面 / Mooncake DRAM 层 / NoF 层) 的网段、适配器与速率矩阵见 § 1.4 网络通路矩阵。

2.4 本地存储

本地 NVMe:

	cx81	cx82
盘	4 × INTEL SSDPF2KX076T1 7.68 TB PCIe (nvme0n1 - nvme3n1)	同型号 4 张
逻辑扇区	512 B	4 KiB
系统盘	sda MR9540-8i 446.6 GB (RAID)	同

2.5 控制机与模型

项	值
控制机	spark-170 (第三台, 只发起 ssh, 不跑引擎)
脚本宿主	/root/pd2/ (只在 cx81 / cx82)
引擎配置目录	/root/dynamo/mooncake-solo-spdk-new/
注入 patch 目录	/root/dynamo/patches-pd/
模型	/raid/models/DeepSeek-V4-Flash-DSpark, 156 GB, 48 分片, 两机均本地 (非共享)
JIT 缓存	/root/cache/dsv4-flash-dspark/ (31 MB: flashinfer / tilelang / vllm)

3. 配置

3.1 容量基线说明

DRAM 的容量在原始记录里有两个分类:

实验	SPDK 的 DRAM	DRAM
20 轮 agent	50 GiB + NoF	400 GiB
Plan A	50 GiB + NoF	400 GiB
Plan B (C=256 锚点)	50 GiB + NoF	400 GiB

3.2 引擎启动参数

```
python3 -m dynamo.vllm \
--model /raid/models/DeepSeek-V4-Flash-DSpark \
--served-model-name DeepSeek-V4-Flash-DSpark \
--tensor-parallel-size 4 \
```

```

--pipeline-parallel-size 1 \
--gpu-memory-utilization 0.90 \
--disaggregation-mode agg \
--kv-transfer-config '{
    "kv_connector": "MooncakeStoreConnector",
    "kv_role": "kv_both",
    "kv_connector_extra_config": {
        "enable_lookup": true,
        "save_decode_cache": false,
        "mooncake_protocol": "rdma",
        "replica_num": 1,
        "nof_replica_num": 0      # ← DRAM 模式 = 0; SPDK 模式 = 1
    }
}' \
--trust-remote-code \
--kv-cache-dtype fp8 \
--block-size 256 \
--prefix-cache-retention-interval None \
--max-num-seqs 32              # 并发上限扩测时为 64
    
```

参数	值	为什么是这个值
--tensor-parallel-size	4	单引擎吃 4 张卡；非 PD 分离。本机无 NVLink，TP=8 时集合通信只跑 11 GB/s（TP=8 端到端只有 1.29×，通信占 91.5%）⇒ TP 不宜超过 4
--gpu-memory-utilization	0.90	0.95 会 OOM
--kv-cache-dtype	fp8	决定 kv_cache_size_tokens = 3,417,941
--block-size	256	cache_config_info.block_size 报 4 是错的，真实命中粒度是 256 token
--prefix-cache-retention-interval	None	★★★ 必须项。不设时 store_mask 用配置值（默认 0 = 只留边界检查点），lookup_mask 却硬编码 None（全部稠密）⇒ 写只写尾巴、查按稠密要，命中恒 0。DSV4 实测：不设时 lookup 1127 / save_put 71（差 23×），设 None 后两侧逐位相同
--max-num-seqs	32（生产） / 64（扩测）	系统并发上限 = 4 × max-num-seqs
save_decode_cache	false	decode 阶段 KV 不落 store ⇒ 写入量由 prefill 唯一决定
replica_num / nof_replica_num	1 / 0 或 1	两种方式的唯一引擎侧变量

3.3 引擎环境变量

变量	值	作用 / 不设的后果
PYTHONHASHSEED	0	★★★ 必须项。不设 ⇒ 跨进程 block hash 不一致 ⇒ 静默 miss（不报错，永远读不到）
MOONCAKE_CONFIG_PATH	/mcfg/vllm.json	引擎侧 mooncake 配置
DYN_SYSTEM_PORT	8101 / 8102 / 8081 / 8082	dynamo 系统端口
ETCD_ENDPOINTS	http://1.1.3.100:2379	服务发现
DYNAMO_STORE_ONLY_PD	0	路由 shim 关闭（聚合模式）
MC_NOF_WORKERS	4	NoF worker 线程数（每个 segment 绑死一个线程）
MC_NOF_SUBMIT_CHUNK_BYTES	131072	单次提交 128 KiB (= 256 块)

变量	值	作用 / 不设的后果
MC_NOF_INFLIGHT_BYTES_LIMIT	33554432	32 MiB 在飞上限 (实测不 binding)
TILELANG_CACHE_DIR	/root/.cache/tilelang	持久化 JIT 缓存, 防高并发中途编译冻结
VLLM_SERVER_DEV_MODE	1	开放 /engine/flush_cache 与 /reset_prefix_cache (测试流程依赖它)
LD_LIBRARY_PATH	/opt/driver:/opt/dynamo/nixl:/opt/dynamo/nixl/plugins	驱动 shim + NIXL
NVIDIA_DRIVER_CAPABILITIES	compute,utility	—
PROMETHEUS_MULTIPROC_DIR	不设	设成空串会让裸 vllm serve 当场死: ValueError: env PROMETHEUS_MULTIPROC_DIR is not set or not a directory (判的是 is not None)。不设才是 vLLM 对裸进程的预期用法

引擎容器的 docker 参数 (RDMA 必需, 缺一项就起不来):

```
--network host \
--device /dev/infiniband \
--ulimit memlock=-1 \
--cap-add IPC_LOCK \
--gpus '"device=0,1,2,3"' # ← 多设备必须内嵌引号, 否则 docker 按逗号切分报错
--shm-size 8g
```

注入式 patch (bind-mount, 非改镜像层):

宿主文件 (/root/dynamo/patches-pd/)	容器内目标
mooncake_connector.py	.../kv_connector/v1/mooncake/mooncake_connector.py
store_scheduler.py	.../mooncake/store/scheduler.py
multi_connector.py	.../kv_connector/v1/multi_connector.py
dynamo_kv_protocols.py	dynamo/vllm/kv_connector_protocols.py

3.4 store 配置

store.json (store 容器读取; local_hostname 必须是 RDMA 字面 IP):

```
{ "local_hostname": "1.1.3.100",
  "metadata_server": "P2PHANDSHAKE",
  "master_server_address": "127.0.0.1:50052",
  "protocol": "rdma",
  "device_name": "mlx5_2",
  "global_segment_size": "200GB",
  "local_buffer_size": "12GB",
  "enable_ssd_offload": false,
  "ssd_offload_path": "/ssd",
  "tenant_id": "default" }
```

字段	cx81	cx82
local_hostname	1.1.3.100	1.1.3.110
master_server_address	127.0.0.1:50052 (本地回环)	1.1.3.100:50052 (跨主机)
device_name	mlx5_2	同
global_segment_size	DRAM "200GB" / SPDK "25GB" (生产 "100GB")	同

字段	cx81	cx82
local_buffer_size	"12GB"	同

vllm.json (引擎侧实际读取, MOONCAKE_CONFIG_PATH=/mcfg/vllm.json) :

```
{ "metadata_server": "P2PHANDSHAKE",
  "master_server_address": "1.1.3.100:50052",
  "protocol": "rdma",
  "device_name": "mlx5_2",
  "mode": "standalone-store",
  "global_segment_size": 0,
  "local_buffer_size": "1GB",
  "enable_offload": true }
```

global_segment_size: 0 $\Rightarrow$ 引擎不提供 DRAM 段, 只做客户端; 段全部由 store 服务进程提供。

enable_offload: true 是 SPDK 初始化的入口 —— 只要为真就会走 SPDK 路径。

kv-aggr.json (引擎参数模板): 内容与 § 1.3.1 所示的连接配置完全一致, 此处不再重复。

3.5 master 配置

```
/usr/local/bin/mooncake_master \
--rpc_port=50052 \
--enable_offload=false \
--offload_on_evict=true \
--promotion_on_hit=true \
--promotion_admission_threshold=1 \
--eviction_high_watermark_ratio=0.9 \
--default_kv_lease_ttl=60s \
--metrics_port=19101
```

★ 实测分配水位停在 85.6% 的来源
★★ 不要调大

--default_kv_lease_ttl 不要调大。上游 scheduler.py:2954 存在单值解包 bug:

```
(req_block_ids,) = self.kv_cache_manager.get_block_ids(req_id)
# ValueError: too many values to unpack (expected 1)
```

hybrid KV 布局使 get_block_ids() 返回多个 group $\Rightarrow$ 单值解包必炸, 打死整个 EngineCore。它只在错误路径触发 (lease 过期 -707 / 传输失败 -800)。调大 TTL 是绕开, 不是修复。

3.6 负载形状 (共 5 种)

负载	输入	输出/轮	轮数	并发 C	命中率	用于
hybsweep.py f×C 阶梯	8192 (32 块)	128	1 (单发)	1...128	$f \in \{0 \dots 1.0\}$	受控介质对比
agent20.py	10,198	128 (自然 EOS)	20	4...256	≈ 0.975	多轮等效性
agent20.py	40,998	128	6	128	—	工作集超容
agent20.py --force-osl	10,198	5,000	20	112	—	长输出崩落
agent20.py --force-osl	10,198	5,000	50	48	—	超长会话

agent 负载的核心构造: 第 t 轮的 messages 里逐字回填前 t-1 轮的 assistant 回复 $\Rightarrow$ 第 t 轮

prompt 以第 t-1 轮为逐字节前缀 $\Rightarrow$ 必然命中。

--force-osl 是必须的: --osl 5000 单独用是空操作 —— 模型约 318 token 就自然 EOS。实测: 不加时每轮 completion_tokens 仍只有 200 - 400; 加上后恰好 5000。

每点唯一随机盐（必须）：/engine/flush_cache 只清本地 APC，清不掉外部 Mooncake store。同一 run-id 复跑时提示字节与上一轮完全相同 $\Rightarrow$ 上一轮写进 store 的 KV 会被直接命中（实测污染可达 97.9%）。

3.7 测量口径

指标	定义	含义
prefill_tok_per_s	$\Delta \text{prefill_kv_computed_tokens} / \Delta \text{prefill_time}$	单请求预填充速度（不随并发放大）
in_tok_per_s	$\Delta \text{prompt_tokens} / \text{wall}$	系统级入口吞吐
decode_tok_per_s	$\Delta \text{generation_tokens} / \Delta \text{decode_time}$	单请求解码速度
out_tok_per_s	$\Delta \text{output_tokens} / \text{wall}$	系统级出口吞吐
tpot_ms	$\Delta \text{decode_time} / \Delta \text{generation_tokens}$	单 token 平均耗时
hit_frac	$\text{prompt_tokens_cached} / \text{prompt_tokens}$	会因上下文增长算术性下降， $\neq$ 失效
超额重算	实际重算 token - 不可避免的新增(osl)	真正的失效度量
停顿判据	generation_tokens_total 增量	绝不用墙钟

3.8 两组负载的健康判据

零复用闸门（用于纯介质对比）：

```
cached_frac == 0.0000%
prompt_tokens_by_source == {local_compute: 100%, local_cache_hit: 0, external_kv_transfer: 0}
```

命中真实性闸门（复用）：

```
load_get_keys > 0 (或 external_kv_transfer > 0)
load_get_failed_keys == 0
master_valid_get_nums_增量 ==  $\Sigma$  store_load_get_keys (两个独立计数器吻合)
master_nof_allocated_bytes_增量  $\approx$  本点写入 (SPDK 模式)
```

完整性闸门（每轮必过）：

```
success == requests
|output_tokens - C  $\times$  osl|  $\leq$  5%
```

errors == 0 不足以证明一轮完整 —— 引擎死亡时连接被切断、客户端读到流自然结束、err 保持 None，会被计入 good。实测有一次 12 个请求返回空流却报 errors=0。

3.9 切模式与核验命令

```
# —— DRAM (400 GiB 纯 DRAM, 两台主机各一次) ——
bash /root/pd2/armprep.sh 200 0      # N = 每机 GiB; 200  $\Rightarrow$  master 400 GiB
bash /root/pd2/engcfg.sh 0           # replica_num=1 / nof_replica_num=0
bash /root/pd2/mnsset.sh 32          # --max-num-seqs
bash /root/pd2/rollout.sh vllm-agg-0 cx81
bash /root/pd2/rollout.sh vllm-agg-1 cx81

# —— SPDK (50 GiB DRAM + NoF 46.09 TB) ——
bash /root/pd2/armprep.sh 25 1
bash /root/pd2/engcfg.sh 1
bash /root/pd2/mnsset.sh 64
bash /root/pd2/rollout.sh vllm-agg-0 cx81
bash /root/pd2/rollout.sh vllm-agg-1 cx81

# —— 收尾：恢复生产（两台主机各一次） ——
```

```
bash /root/pd2/planB_restore.sh
```

每次切模式后必须核验：

```
# ① 容量对不对
curl -s http://127.0.0.1:19101/metrics | grep -E '^master_(total_capacity|nof_allocated)_bytes'
#   DRAM 400 GiB → 429496729600 ; SPDK → 53687091200

# ② 配置真的进了容器 argv (不要从脚本读)
docker inspect vllm-agg-0 --format '{{json .Args}}' | python3 -c "
import re,sys,json
blob=' '.join(x for x in json.load(sys.stdin) if isinstance(x,str))
print('max-num-seqs    =', (re.findall(r'max-num-seqs (\d+)',blob) or ['?'])[0])
print('nof_replica_num =', (re.findall(r'nof_replica_num\D{0,5}(\d+)',blob) or ['?'])[0])
# △ 不要用 r'replica_num\D{0,5}(\d+)' —— 会命中 nof_replica_num 的尾巴
"

# ③ 亲和性 (若用未修复镜像)
for k in 0 1 2 3: do
    W=$(python3 /root/pd2/findpid.py $(docker inspect -f '{{.State.Pid}}' vllm-agg-0) VLLM:Worker_TP$k | head
-1)
    echo "TP$k $(awk '/Cpus_allowed_list/{print $2}' /proc/$W/status)"
done
```

4. 性能对比

4.1 纯读路径

f=1.0 (全复用, 命中读全部由 store 供给)

C	SPDK 出 tok/s	DRAM 出 tok/s	DRAM/SPDK
1	42.9	42.9	1.000
8	153.2	155.4	1.014
16	229.3	225.0	0.981
32	294.8	290.5	0.985
64	380.0	372.2	0.980

吞吐比 0.97 - 1.03 ⇒ 打平 (噪声内), 其中 C=8 一档 SPDK 略高。这独立复现了既有结论: DRAM 与 NoF 在读路径上的差距在 0.7 - 10% 之间。

4.2 对比的有效性

这是本报告所有对比成立的前提: 两模式的外部 KV 供给量完全相同, 所以差异只可能来自「字节落在哪个介质」。

量	SPDK	DRAM	差
save_put 总 keys	1,483,040	1,484,463	+0.1%
load_get 总 keys	308,672	309,732	+0.3%
设计命中率	0.4849 / 0.7274 / 0.9701	同	完全相同
总错误	0	0	—

量	SPDK	DRAM
master_nof_allocated_bytes 全程增量	+1,486,362,009,600 B (1.486 TB)	0.0 B
逐格 NoF 增量最大值	182,936,862,720 B	0.0
master_allocated_bytes 全程增量	+26,144,432,640 B (50 GiB 环, 淘汰后净值)	+163,111,553,280 B (400 GiB 池)
master_total_capacity_bytes	53,687,091,200 (50 GiB)	429,496,729,600 (400 GiB)

DRAM 方案写了约 1.49 TB 的对象，一个字节都没落到 NoF 上。介质身份由机器判据确认。

SPDK 模式的两层确实都被写入：

格子	master_allocated_bytes Δ (DRAM)	master_nof_allocated_bytes Δ (NoF)
f=0 C=1	+22,867,107,840	+22,867,107,840
f=0 C=4	-765,711,360 (淘汰 > 新增)	+22,867,107,840
f=1.0 C=32	0	0 (复热池已全在)
模式结束时水位	46,153,152,000 / 53,687,091,200 (46.15 GB / 50 GiB)	2,097,826,629,120

4.3 20 轮 agent 会话 (osl=128, C=4...128)

该实验 DRAM = 400 GiB (armprep.sh 200 0)，SPDK = 50 GiB + NoF 46.09 TB。两模式稳态命中率均 ≈ 0.975 ，14 个点全部 0 错误。

模式	C=4	C=16	C=32	C=64	C=96	C=128
DRAM 稳态 RPS	2.97	7.19	8.93	8.79	11.20	17.33
SPDK 稳态 RPS	2.91	7.01	8.75	8.36	10.66	17.31
差异 (SPDK 相对 DRAM)	-2.0%	-2.5%	-2.0%	-4.9%	-4.8%	-0.1%
DRAM 稳态出 tok/s	380.1	920.3	1138.2	1121.5	1425.7	2201.9
SPDK 稳态出 tok/s	349.0	896.8	1120.4	1069.4	1362.6	2208.5
tok/s 差异	-8.2%	-2.6%	-1.6%	-4.6%	-4.4%	+0.3%

解码路径 (TPOT)：

C	4	16	32	64	96	128
DRAM TPOT p50 ms	8.85	13.13	21.91	46.00	49.14	39.08
SPDK TPOT p50 ms	8.90	13.24	21.15	43.42	46.61	39.55
差异	+0.6%	+0.8%	-3.5%	-5.6%	-5.1%	+1.2%

⇒ 解码路径 (占每轮墙钟 70%) 不受介质影响：6 档里 4 档 SPDK 略优，最大相对差 5.6%。

容量与两 tier 占用：

	C=32 t19	C=64 t19	C=96 t19	C=128 t1	C=128 t19
SPDK • DRAM	47.4 GB	48.3 (锁死)	48.3 (锁死)	47.9 GB	48.3 (锁死)
SPDK • NoF	2,169.5 GB	2,244.0	2,356.1	2,473.3	2,505.7

	C=32 t19	C=64 t19	C=96 t19	C=128 t1	C=128 t19
DRAM • DRAM	71.7 GB	146.1	258.2	375.4	369.1 (开始驱逐)
DRAM • NoF	2,097.8 (不动)	2,097.8	2,097.8	2,097.8	2,097.8

命中来源拆解 (稳态, 占 prompt token 比例) :

模式	本地重算	本地 APC 命中	外部 store 读	外部读 keys/轮
SPDK @C=64	0.0249	0.8880	0.0803	4,840
SPDK @C=128	0.0252	0.9149	0.0827	9,424
DRAM @C=128	0.0250	0.8730	0.1135	15,880

外部 store 只贡献 8-12% 的 prompt token, 其余 87-91% 由引擎自己的 GPU APC 提供, 重算只占 2.5%。⇒ 三层协同 (GPU APC → DRAM 段 → NoF) 把命中供给分散到三层, store 层从未被逼成瓶颈 —— 这是吞吐打平的机理。

4.4 更高并发 (C=256, 20 轮)

把 --max-num-seqs 从 32 提到 64 (系统上限 $4 \times 64 = 256$) 后, 同负载形态的三点对照:

模式	有效容量	稳态 RPS	每槽 RPS	TPOT p50	命中率	点累计写入
SPDK KV Cache (50 GiB + NoF 46.09 TB)	46.1 TB	25.67	0.1003	51.30	0.9750	300.5 GB
DRAM KV Cache (400 GiB 纯 DRAM)	429.5 GB	24.27	0.0948	53.38	0.9752	300.3 GB

SPDK 反超 +5.5%, 命中率逐位相同 (0.9750 vs 0.9752), store 工作量相同 (300.5 vs 300.3 GB)。⇒ C=256 处的饱和是引擎侧的, 不是 store 侧的。把 DRAM 段从 50 GiB 扩到 400 GiB (8×)、完全不用 NoF, 并不能提升吞吐。

SPDK 模式的并发阶梯 (mns=64) :

C	稳态 RPS	每槽 RPS	出 tok/s	TPOT p50	命中率
64	8.30	0.1297	1057.8	48.04	0.9751
128	16.69	0.1304	2131.4	39.76	0.9749
160	20.64	0.1290	2629.7	40.55	0.9751
192	21.80	0.1136	2778.8	43.49	0.9752
224	24.02	0.1072	3052.3	45.72	0.9752
256	25.67	0.1003	3258.7	51.30	0.9750

每槽 RPS 在 C=64→160 完全平稳 (0.1290 - 0.1304, 抖动 <1%), 拐点在 C≈160, 之后降到 C=256 的 0.1003。NoF 全阶梯累计写入 1202.6 GB = 46.09 TB 的 2.61% —— 容量远未触顶。

4.5 每轮墙钟的加法

以 SPDK 模式 C=128 / t19 为例:

每轮墙钟	7.39 s	
TTFT p50	2.06 s (28%)	← store 读取发生在这里
解码 128 tok × TPOT	40.3 ms = 5.16 s (70%)	
两者之和	7.22 s ≈ 墙钟 7.39 s	⇒ 几乎没有 "store 等待" 这一项

结论：每轮的时间几乎全部是「TTFT + 解码」，没有额外的 store 阻塞项。KV 读取发生在 TTFT 里，而 TTFT 只占墙钟的 25 - 28%；解码占 70%。⇒ KV 读取路径的差异被解码时间稀释，这就是吞吐打平的物理原因。

第三层存储（本报告识别出）：GPU 自带的 prefix cache：

GPU APC: 4 引擎 × 3,471,060 token = 13.88M token

它在 20 轮负载下承担 87 - 91% 的命中 token（见 § 4.3 表），进一步压缩了 store 层差异的可见度。

4.6 批次形状效应（两模式同时出现，与介质无关）

每引擎批	1	2	4	8	16	24	32
TPOT p50 ms	8.85	10.30	13.13	21.91	46.00	49.14	39.08

满批（32 = max-num-seqs）反而比 24 更快 ⇒ CUDA graph 按批大小捕获。因此 C=96（每引擎 24）比 C=128（每引擎 32）慢 24% 是真实的、可复现的，两模式同时出现。⇒ 调并发应优先落在「每引擎满批」的点上（C 取 4 的整数倍且 C/4 整除 32）。

5. 多轮对话

5.1 容量：107×

	DRAM KV Cache	SPDK KV Cache
有效容量	429.5 GB	46.09 TB
等效 token 容量	≈ 4.8M token	≈ 1092M token
相当于多少轮 osl=5000 的写入（C=48）	~20 轮	~2,140 轮

容量规划公式（已在实测中验证）：

t_1 突发 = $C \times \text{prompt_base} \times 88.1 \text{ KB}$

$t \geq 2$ 每轮增量 = $C \times \text{osl} \times 88.1 \text{ KB}$

工作集(t) = $C \times [\text{prompt_base} + (t-1) \times \text{osl}] \times 88.1 \text{ KB}$

耗尽轮次 $t^* = 429.5 \text{ GB} / (C \times \text{osl} \times 88.1 \text{ KB})$ （崩坏再滞后 1~2 轮）

验证：C=112 / osl=5000 预测 8.6 轮 → 实测正好 t8 崩坏。

DRAM 的失效不是「缓降」而是「悬崖」，且不可逆 —— 因为 vLLM 前缀缓存是链式块哈希（第 k 块哈希含第 k-1 块），LRU 驱逐的最旧的块正好是每条会话的链头；链头一掉，整条会话的复用一起失效。叠加正反馈：驱逐 → 重算 → 重写 → 更多驱逐。实测单轮写入从 50 GB 飙到 1,051.8 GB。

5.2 20 轮

量	DRAM 模式	SPDK 模式
C=128 工作集	≈156 GB	≈156 GB
DRAM 段实际水位	C=128 t19 369.1 GB（开始驱逐）	C=32 起锁死 48.3 GB / 53.7 GB
⇒ DRAM 能装下工作集的比例	100%	仅 ~1/3
稳态命中率	0.9751	0.9752

SPDK 在 DRAM 只能装下 1/3 工作集的情况下，命中率仍是 0.9752 $\Rightarrow$ 至少 2/3 的命中读由 NoF 服务。「50 GiB DRAM 能撑住 97.5% 命中」这件事本身，就是 NoF 在兜底的证明——若没有 NoF，50 GiB 的 DRAM 在 $C \geq 32$ 命中率必然崩塌。直接证据：master_nof_allocated_bytes 随 C 单调增长（ $C=128$ 单点 +2.90 GB），而 DRAM 模式同一指标全程 0 增量。

5.3 工作集超出 DRAM 容量

参数：C=128、文档 40,998 token、6 轮、终态 41,758 token/会话 $\Rightarrow$ 工作集 471 GB DRAM 模式容量 400 GiB = 429.5 GB（实测均衡只到 383.3 GB） $\Rightarrow$ 缺口 88 GB $\approx$ 19%

轮	DRAM 墙钟 s	DRAM 命中率	DRAM 重算 token	SPDK 墙钟 s	SPDK 命中率	SPDK 重算 token
t1 (冷)	207.4	0.0000	5,247,744	214.3	0.0000	5,247,744
t2	121.0	0.4821	2,727,765	11.4	0.9954	24,133
t3	121.0	0.6353	1,927,976	10.4	0.9918	43,369
t4	25.2	0.9313	364,698	10.4	0.9943	30,002
t5	10.1	0.9906	50,215	10.6	0.9907	49,494
t6	9.6	0.9931	37,041	11.5	0.9932	36,253

t2：墙钟差 10.6 $\times$ ，命中率 0.4821 vs 0.9954，重算量差 113 $\times$ 。t2 - t4 是容量崩溃的代价，共重算 502 万 token，多花 267 s ($\approx$ 4.5 分钟)。t5 - t6 两模式收敛到同一水平（命中率 0.9906/0.9907、0.9931/0.9932，墙钟 10.1/10.6、9.6/11.5）——崩溃过去之后，两模式行为一致。

SPDK 模式的 tier 占用（逐轮 master 差分）：

轮	master_dram_delta	master_nof_delta	累计 NoF 增量
t1	+46.0 GB	+472.1 GB	472.1 GB
t2	0	0	472.1 GB
t3	-1.9 GB	+3.0 GB	475.1 GB
t6	+0.7 GB	+2.5 GB	480.6 GB

t1 的 master_nof_delta = 472.1 GB 恰好等于工作集大小（5,247,744 token $\times$ 88.1 KB = 472.1 GB）——这是 NoF 承接了整个工作集的直接证据。

■ 口径说明：DRAM 模式在 t5 起「自愈」到 0.99，是因为引擎自带的 GPU 前缀缓存（4 $\times$ 3,471,060 = 13.88M token）大于工作集 5.345M token，在本次独占 GPU 的条件下把它兜住了。真实部署中 GPU 会被其它流量冲刷，不会有这种余量，因此该「自愈」不可外推。

5.4 长输出 (osl=5000)：112 $\times$

--osl 5000 单独用是空操作（模型 $\sim$ 318 token 即 EOS），必须加 ignore_eos。C 的选择由引擎自身 KV cache 卡死： $(C/4) \times 110,199 \leq 3,417,941 \Rightarrow C \leq 124$ ，取 C=112（每引擎 28 序列 = 90.3%，不触发抢占）。

轮	DRAM 命中率	DRAM 超额重算/请求	DRAM TTFT p50	SPDK 命中率	SPDK 超额重算/请求	SPDK TTFT p50
2	0.6577	195	12,360	0.6590	167	11,990
4	0.7975	105	12,682	0.7978	96	12,359
6	0.8546	127	13,910	0.8546	127	12,951
7	0.8676	336	13,346	0.8721	150	13,253

轮	DRAM 命中率	DRAM 超额重算/请求	DRAM TTFT p50	SPDK 命中率	SPDK 超额重算/请求	SPDK TTFT p50
8	0.7526	6,213	18,377	0.8856	183	13,883
10	0.8274	4,560	30,220	0.9058	214	14,317
14	0.7529	13,656	32,739	0.9317	151	14,913
16	0.5181	36,223	50,474	0.9394	185	16,150
18	0.0169	88,979	215,938	0.9463	131	16,658
20	0.0085	99,754	246,236	0.9516	117	16,852

t2 - t7 两模式几乎完全重合（命中率 0.658/0.659、0.7457/0.7451、0.7975/0.7978、0.8285/0.8283...）⇒ 前 7 轮根本没到容量边界，两模式的行为是同一套；分叉精确发生在 DRAM 写满的那一轮。

汇总：

指标	DRAM KV Cache	SPDK KV Cache	比值
点总墙钟	7,480 s	5,036 s	1.49×
t20 命中率	0.0085	0.9516	112×
t20 超额重算/请求	99,754	117	853×
t20 TTFT p50	246,236 ms	16,852 ms	14.6×
t20 单轮墙钟	727.3 s	277.7 s	2.62×
累计写入	5,544,909 keys / 5,557.3 GB	1,061,099 keys / 1,063.5 GB	5.2×
累计外部读	1,106,248 keys	1,808,452 keys / 1,812.5 GB	0.61×

t20 的解析上限：可复用前缀 100,624 / prompt 105,624 ⇒ 理想 hit_frac = 0.9527。⇒ SPDK 实测 0.9516 = 上限的 99.88%；DRAM 实测 0.0085 = 上限的 0.9%。SPDK 模式的超额重算/请求 全程 96 - 214 token（占可复用前缀 0.12% - 1.64%，且逐轮收敛），说明随着会话变长，缓存反而越来越完整。

DRAM Only 崩落的三条机制证据：

1. master_dram_delta 转负：t1 - t6 累计 +354.45 GB，t7 只 +13.06 GB，t8 起为负（-5.51、-11.43、-10.21...）⇒ 分配峰值 $\approx 367.5 \text{ GB} = 429.5 \text{ GB}$ 的 85.6%，此后转入 LRU 驱逐（与 --eviction_high_watermark_ratio=0.9 吻合）
2. 写入量被崩落放大 20 倍：t2 - t6 稳定 50 GB/轮 → t16 410.6 → t18 946.3 → t20 1,051.8 GB
3. t16→t20 是悬崖不是斜坡：0.5181 → 0.1758 → 0.0169 → 0.0085 → 0.0085，外部读塌到 898 token（ ≈ 3.5 个 256-token 块）

SPDK 模式无崩落的三条机制证据：

1. 读总量 = 段容量的 33.8 倍：1,812.5 GB 读 vs 53.69 GB DRAM 段
2. 单轮读量已超段容量：t10 87.35 GB（1.63×）、t15 128.22 GB（2.39×）、t20 171.58 GB（3.20×）

3. 两段增量走势分叉: master_dram_delta 从 t3 起在 ± 2 GB 内摆动 (饱和、边写边驱逐);
master_nof_delta 全程恒为正 ~ 50.9 GB/轮 (46 TB 从不驱逐) 交叉核对 (强证据): master 自身 valid_get_nums_ 增量 +1,808,452, 与引擎上报的 store_load_get_keys 合计精确相等;
master_nof_allocated_bytes 增量 +1,063.5 GB 与全点写入量精确吻合。

5.5 50 轮超长会话

量	值
kv_cache_size_tokens/引擎 $\times 4$	$3,471,060 \times 4 = 13,884,240$
prompt 增长律 (实测拟合)	$\text{prompt}_t \approx 10,204 + (t-1) \times 5,022$
t50 每请求 prompt	$\approx 256,282$
C=48 时 t50 常驻需求	$48 \times 261,282 = 12,541,536 = 90.3\% \quad \checkmark$
C=112 时 t50 常驻需求	$112 \times 261,282 = 29,263,584 = 211\% \quad \times$ 自 t23 起抢占

轮	墙钟 s	命中率	重算/请求	超额重算	超额占前缀	TPOT p50
2	264.55	0.6580	5,190	190	1.87%	46.89
10	274.63	0.9057	5,222	222	0.44%	49.10
20	286.61	0.9515	5,118	118	0.12%	46.27
30	279.83	0.9668	5,177	177	0.12%	49.56
40	286.72	0.9752	5,118	118	0.06%	47.88
50	305.85	0.9797	5,169	169	0.07%	53.65

全点合计: 写入 1,098,006 keys / 1,024.89 GiB; 读取 3,588,036 keys / 3,349.10 GiB; 外部取回 219,733,248 token; 输出 11,866,835 token; 点墙钟 14,047.91 s = 3.90 h, 0 错误。

两个 tier 的角色被 50 轮坐实: nof Δ 自 t2 起每轮恒为正 +20.4 GiB (50 轮一次不落); dram Δ 在 $\pm 1 - 3$ GiB 震荡 $\Rightarrow$ DRAM 段早在 t2 就饱和并持续驱逐, 容量全部由 NoF 承担。NoF 46.09 TB $\div$ 21.5 GB/轮 $\approx$ 2,140 轮才填满。

对照: 同样 osl=5000 的 C=48 负载, DRAM 模式 400 GiB 大约在 t20 就耗尽 ($1 + (101,554 - 10,200)/5,022 = 19.3$) —— 50 轮会话用纯 DRAM 在算术上不可行。与 20 轮点 (C=112) 前 20 轮同轮次 hit_frac 逐点差全部 $\leq 0.001 \Rightarrow$ 并发 C 不改变 KV 复用行为, 命中率完全由「会话轮次」这一维决定。

5.6 收益的来源

SPDK 是不会驱逐的容量。

容量轴: 429.5 GB $\xrightarrow{\quad\quad\quad}$ 46.09 TB (107 $\times$)
└─ 多轮长会话从「崩落」变为「命中率单调上升」

读路径: 吞吐 0.97 - 1.03 $\times$ (打平) $\leftarrow$ 固定开销主导, 介质差异被解码时间稀释
解码: TPOT 一致 $\leftarrow$ 占每轮墙钟 70%, 不受介质影响

为什么读路径能打平: 这条路径是 固定开销 (≈ 1 s) + 字节 $\div$ 带宽 的结构, 对长度不敏感。介质参与的是「字节 $\div$ 带宽」那一项, 而它本来就不是大头; 且读取发生在只占墙钟 25 - 28% 的 TTFT 里。

为什么容量收益是决定性的：DRAM 的失效是链式块哈希 + LRU 驱逐链头造成的悬崖，而 NoF 提供的是一个不会驱逐的落点。

6. 结论

- SPDK KV Cache 在性能上与 DRAM 打平。
 - 纯读路径吞吐 $0.97 - 1.03 \times$ （噪声内，C=8 一档 SPDK 略高）
 - 20 轮稳态 RPS 差 $-0.1\% \sim -4.9\%$ ，C=128 时只差 0.1%；出 tok/s 在 C=128 时 SPDK +0.3%
 - C=256 更高并发下 SPDK 反超 +5.5% (25.67 vs 24.27 RPS，命中率逐位相同)
 - 解码路径 (TPOT) 两模式基本一致，6 档里 4 档 SPDK 略优 —— 占每轮墙钟 70% 的解码不受介质影响
- 对比的前提是 SPDK 模式的 DRAM 只有 50 GiB (DRAM 模式的 1/8)。它在 DRAM 侧先让了 8 倍，仍然在吞吐上追平乃至反超。
- 对比的有效性有机器判据：两模式 save_put 差 +0.1%、load_get 差 +0.3%；DRAM 模式写了 1.49 TB 对象且 master_nof_allocated_bytes 增量为 0.0 B；SPDK 模式的 DRAM 与 NoF 两层都被真实写入 (f=0/C=1 各 +22,867,107,840)。
- 多轮对话的收益是决定性的：
 - 容量 $107 \times$ (429.5 GB $\rightarrow$ 46.09 TB)
 - 20 轮：50 GiB DRAM 只能装下 1/3 工作集， $\geq 2/3$ 的命中读由 NoF 服务，命中率仍 0.9752
 - 工作集超容 (471 GB)：t2 墙钟 121.0 s $\rightarrow$ 11.4 s ($10.6 \times$)，命中率 0.4821 $\rightarrow$ 0.9954，重算量 $113 \times$
 - 长输出 (osl=5000)：t20 命中率 0.0085 $\rightarrow$ 0.9516 ($112 \times$)，单轮墙钟 727.3 s $\rightarrow$ 277.7 s ($2.62 \times$)，TTFT 246 s $\rightarrow$ 16.9 s ($14.6 \times$)
 - 50 轮超长会话：命中率 0.6580 $\rightarrow$ 0.9797 单调上升，超额重算长期维持在可复用前缀的 0.06 - 0.2%，无崩落；同等负载下 DRAM 模式在 t20 就耗尽，50 轮在算术上不可行。
- DRAM 模式的失效是「悬崖」而不是「缓降」，且不可逆 —— 链式块哈希 + LRU 驱逐链头 $\Rightarrow$ 整条会话复用一起失效；叠加「驱逐 $\rightarrow$ 重算 $\rightarrow$ 重写」正反馈（单轮写入从 50 GB 飙到 1,051.8 GB）。崩落轮次可在测试前精确预测 ($429.5 \text{ GB} / (C \times \text{osl} \times 89.4 \text{ KB}) = 8.6 \text{ 轮} \rightarrow \text{实测 t8}$)。
- SPDK 模式的命中质量更高：长输出场景下 t20 命中率 0.9516 = 解析上限的 99.88%，超额重算/请求 全程 96 - 214 token 且逐轮收敛 (0.12% - 1.64% $\rightarrow$ 0.12%)。

	DRAM KV Cache	绿算 GP 7000 SPDK KV Cache
KV 落点	Mooncake DRAM 段 (纯内存)	DRAM 段 + NVMe-oF (SPDK) 副本层
引擎开关	replica_num: 1 / nof_replica_num: 0	replica_num: 1 / nof_replica_num: 1
store 配置	global_segment_size: "200GB"/机 $\Rightarrow$ 400 GiB	global_segment_size: "25GB"/机 $\Rightarrow$ 50 GiB
有效容量	429.5 GB	46.09 TB

SPDK 的 DRAM 段只有 50 GiB，是 DRAM (400 GiB) 的 1/8。它不是「同样大小的 DRAM 再加一层 NoF」，而是「DRAM 砍到 1/8，靠 NoF 把容量补回来」。因此下面的每一个性能结论都建立在 SPDK 模式先在 DRAM 侧让了 8 倍的前提下。

6.1 收益

维度	DRAM KV Cache	绿算 GP 7000 SPDK KV Cache	结论
纯读路径吞吐 (f=1.0, C≤64)	基准	0.97 - 1.03×	打平 (噪声内)
20 轮稳态 RPS (os1=128, C=4...128)	基准	-0.1% ~ -4.9%	差异在噪声量级
C=256 更高并发 (20 轮)	24.27 RPS	25.67 RPS	SPDK 反超 +5.5%
20 轮稳态 TPOT (解码单 token)	基准	基本一致	解码路径不受介质影响
长会话命中率 (os1=5000, 20 轮)	0.0085 (崩落)	0.9516	112×
长会话单轮墙钟	727.3 s	277.7 s	2.62×
工作集超容 (471 GB)	t2 墙钟 121.0 s	11.4 s	10.6×
50 轮会话	不可行	命中率 0.6580 → 0.9797 单调升	无崩落
有效容量	429.5 GB	46.09 TB	107×

SPDK KV Cache 在吞吐上与 DRAM 打平 (纯读 0.97 - 1.03×, 高并发 C=256 下反超 5.5%), 同时把有效容量做到 107 倍, 使多轮长会话从「崩落」变为「命中率单调上升」。



E-Mail: ls@luisuantech.com

联系电话: 010-52129203/010-52129206

运营中心: 北京市海淀区西直门北大街甲43号金运大厦A座803室

本文件内容解释权归绿算技术有限公司所有