Penguin Solutions MemoryAI™ KV Cache Server Software Installation Guide

Penguin Solutions
R807-00

Notice

The information contained in this document is subject to change without notice.

UNLESS EXPRESSLY SET FORTH IN A WRITTEN AGREEMENT SIGNED BY AN AUTHORIZED REPRESENTATIVE OF PENGUIN SOLUTIONS, PENGUIN SOLUTIONS MAKES NO WARRANTY OR REPRESENTATION OF ANY KIND WITH RESPECT TO THE INFORMATION CONTAINED HEREIN, INCLUDING WARRANTY OF MERCHANTABILITY AND FITNESS FOR A PURPOSE.

Penguin Solutions assumes no responsibility or obligation of any kind for any errors contained herein or in connection with the furnishing, performance, or use of this document. Software described in Penguin Solutions' documents (a) is the property of Penguin Solutions (PENG) Ireland Limited or the third party, (b) is furnished only under license, and (c) may be copied or used only as expressly permitted under the terms of the license.

Penguin Solutions' documentation describes all supported features of the user interfaces and the application programming interfaces (API) developed by Penguin Solutions. Any undocumented features of these interfaces are intended solely for use by Penguin Solutions personnel and are subject to change without warning.

This document is protected by copyright. All rights are reserved. Penguin Solutions grants you limited permission to download and print a reasonable number of copies of this document (or any portions thereof), without change, for your internal use only, provided you retain all copyright notices and other restrictive legends and/or notices appearing in the copied document.

Penguin Solutions is a registered trademark of Penguin Solutions. MemoryAI is a trademark of Penguin Solutions.

All other trademarks are the property of their respective owners.

Manual Name: *Penguin Solutions MemoryAI KV Cache Server Software Installation Guide*

Part Number: R807
Revision Number: 00
Publication Date: October 2026

Penguin Solutions
5 Mill and Main Place, Suite 500
Maynard, Massachusetts 01754-2660

© 2026 Penguin Solutions (PENG) Ireland Limited. All rights reserved.

Contents

Preface vi

1. Installing and Configuring Penguin Solutions MemoryAI KV Cache Server 1-1
- Understanding Product Architecture 1-3
  - Key Architectural Boundaries 1-4
- Components and Software Configurations 1-4
- Deployment Prerequisites and Network Specifications 1-8
- Installing and Configuring Penguin Solutions MemoryAI KV Cache Server 1-9
  - Deploying the CXL KV Storage Server 1-9
    - Download the Repository and Packages 1-10
    - Prepare the Ubuntu Environment 1-10
    - Install Penguin Velocity 9.1.4 1-11
    - Configure the Network Bind Address and Ports 1-11
    - Examine Memory Nodes and CPU Ownership 1-12
    - Start the Eight-Primary Cluster 1-12
    - Validate the CXL KV Storage Server Deployment 1-13
    - Perform Operational Checks 1-13
    - Troubleshooting the CXL KV Storage Server Deployment 1-15
    - Performing a Planned Shutdown 1-16
    - Modifying a Configuration 1-16
    - Changing the Number of Primaries 1-17
  - Deploying the GPU Inference Server 1-17
    - Confirm the GPU Inference Server Prerequisites 1-18
    - Extract the Supplied Inference Server `.zip` File 1-18
    - Install the Ubuntu, NVIDIA, and Pinned Python Packages 1-19
    - Install the Selected KV-Cache Transport 1-19
    - Install the CUDA 12.9-Compatible LMCache `c_ops` Extension 1-21
    - Configure `site.env` and Render the LMCache `.yaml` File 1-22
    - Provision Model Weights 1-23

Start Inference and Verify Service 1-24
Validate the External KV Path 1-25
Troubleshooting the GPU Inference Server Deployment 1-26
Configuring and Launching Penguin Solutions MemoryAI KV Cache Server 1-26
Configure the LMCache RDMA `.yaml` file 1-27
Launch the vLLM Inference Engine 1-27
Deploying the AIPerf Load Generator 1-27
Confirm the AIPerf Load Generator Prerequisites 1-28
Extract the `.zip` File 1-29
Install the AIPerf and Ubuntu Prerequisites 1-29
Configure the GPU Inference Server Endpoint 1-30
Verify the Load Generator and API 1-31
Run a One-Request Smoke Benchmark 1-32
Run a Measured Workload 1-33
Locate and Review Benchmark Results 1-33
Troubleshooting the AIPerf Load Generator Deployment 1-34
Verifying and Troubleshooting 1-34
Verifying Server Health 1-34
Reverting from RDMA to GLIDE TCP Mode 1-35
Enforcing LRU Memory Eviction 1-35

Figures

Figure 1-1. Deployment Topology 1-3

Tables

Table 1-1.	Key Concepts	1-1
Table 1-2.	CXL KV Storage Server (`cxL-box`) Components	1-4
Table 1-3.	GPU Inference Server (`gpu-serve`) Components	1-6
Table 1-4.	AIPerf Load Generator (`gpu-Loadgen`) Components	1-8
Table 1-5.	Network Configuration Example	1-9
Table 1-6.	Operational Checks	1-13
Table 1-7.	Troubleshooting CXL KV Storage Server Deployment	1-15
Table 1-8.	GPU Inference Server `site.env` Configuration Settings	1-22
Table 1-9.	Key LMCache Configuration Settings for TCP and RDMA	1-23
Table 1-10.	Troubleshooting GPU Inference Server Deployment	1-26
Table 1-11.	Workload Settings in the `load-gen.env` File	1-30
Table 1-12.	Troubleshooting the AIPerf Load Generator Deployment	1-34
Table 1-13.	Verifying Server Health	1-35

Preface

The *Penguin Solutions MemoryAI KV Cache Server Software Installation Guide* (R807) documents the deployment procedures for installing and configuring Penguin Solutions MemoryAI KV Cache Server.

Revision Information

This is the first edition of this document.

Typographical Conventions

This manual uses the following typographical conventions:

- **Boldface** emphasizes words in text. For example:

  Install the KV-cache transport using **one** of the following methods.

- `Monospace` represents text that would appear on your screen (such as commands, code fragments, and names of files and directories). For example:

  Do not run `apply-site.sh` or manually run `lib-bind.sh`.

- *`Monospace italic`* represents terms that are to be replaced by literal values. In the following example, the user must replace the monospace-italic term with a literal value.

  `KV_SERVER_HOST="`*`actual_CXL_server_HCA_IP`*`"`

Chapter 1
Installing and Configuring Penguin Solutions MemoryAI KV Cache Server

This manual provides the deployment procedures for installing and configuring Penguin Solutions MemoryAI KV Cache Server. It discusses the following topics:

- “Understanding Product Architecture” on page 1-3
- “Components and Software Configurations” on page 1-4
- “Deployment Prerequisites and Network Specifications” on page 1-8
- “Installing and Configuring Penguin Solutions MemoryAI KV Cache Server” on page 1-9
- “Verifying and Troubleshooting” on page 1-34

Table 1-1 defines some key concepts discussed in this manual.

Table 1-1. Key Concepts

Concept	Definition
AIPerf Load Generator	Runs AIPerf 0.5.0 and sends OpenAI-compatible requests to the GPU Inference Server.
CXL® KV Storage Server	Runs an eight-primary Valkey Cluster using Penguin Velocity server software, with KV-cache data stored in CXL memory.
CXL Type-3 memory	Memory expansion attached through Compute Express Link.
Dataset	A collection of data used to train, test, or evaluate an AI model.
Eight-primary Valkey cluster	Consists of eight Valkey server instances running on one physical CXL KV Storage Server. Each primary manages a portion of the KV-cache data stored in CXL memory.
GLIDE client	General Language Independent Driver for the Enterprise client. It is an official open-source client library for Valkey.

Table 1-1. Key Concepts *(Continued)*

Concept	Definition
GPUDirect RDMA	NVIDIA® technology enabling direct data exchange between GPUs and peer devices such as network adapters.
GPU Inference Server	Runs vLLM/LMCache, exposes the AI-compatible/v1 API (reference port 8000), and processes requests.
HCA	Host channel adapter. The HCA connects the host to a high-speed fabric such as InfiniBand and supports high-performance communication such as RDMA.
Hugging Face®	An AI platform and community where developers can find, share, test, and work with AI models and datasets.
InfiniBand®	High-performance interconnect that can carry RDMA traffic.
KV cache	Temporary key-value data produced while an AI model processes prompts. Reusing it can reduce repeated work.
LMCache	The software layer that managed reusable KV cache and connects vLLM to the remote Valkey back-end.
`load-gen.env` file	A configuration file that stores environment variables and keeps sensitive data out of source code.
NIC	Network interface card.
NUMA	A server memory layout in which access speed depends on which processor memory node is used. The launcher binds Valkey processes across CXL NUMA nodes 2 and 3.
RDMA	Remote Direct Memory Access. It moves data between systems with low CPU overhead. Data lands in pinned host memory on the GPU server, not directly in GPU VRAM.
RoCE	RDMA over Converged Ethernet.
Shim	Sits between two software components and intercepts or redirects calls so the components can work together without requiring major changes.
Smoke test	A quick, basic test that checks whether an application, model, agent, or pipeline is working well enough for more thorough testing.
Tensor parallelism	The model workload is split across eight GPUs. “TP=8” means that all eight GPUs participate.

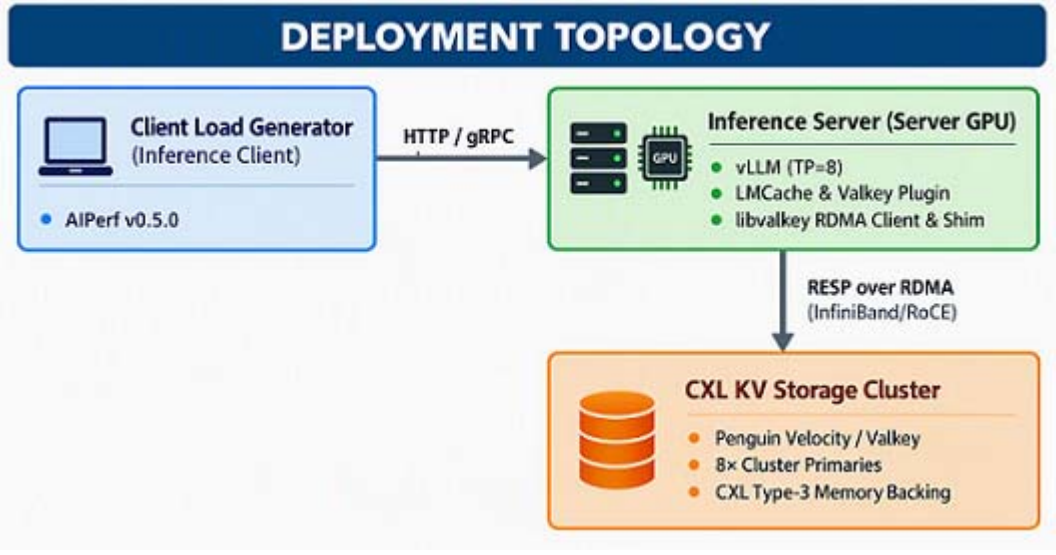
Table 1-1. Key Concepts *(Continued)*

Concept	Definition
Valkey	A high-performance, open-source key/value datastore.
Valkey Cluster	The logical group of eight primary instances running on the CXL KV Storage Server.
vLLM	Virtual Large Language Model. The model-serving engine that exposes an OpenAI-compatible API endpoint.

Understanding Product Architecture

The architecture of Penguin Solutions MemoryAI KV Cache Server extends KV-cache storage beyond GPU memory using CXL Type-3 memory on a dedicated cache server. GPU Inference Servers running vLLM access this external cache through LMCache over an RDMA network using InfiniBand or RoCE. When previously computed KV-cache data can be reused, retrieval from the external cache can reduce redundant computation and improve inference efficiency. Figure 1-1 illustrates the deployment topology.

Figure 1-1. Deployment Topology



Key Architectural Boundaries

The key architectural boundaries are as follows:

- Three-host separation: To ensure predictable latency and prevent resource contention, the CXL KV Storage Server, the GPU Inference Server, and the AIPerf Load Generator must be deployed on separate physical servers.
- RDMA host-memory boundary: Valkey RDMA transfers payloads directly into pinned host-system memory on the GPU Inference Server. Payloads are not written directly into GPU VRAM; instead, they are written into the non-GPUDirect RDMA path.
- Dedicated inference role: The GPU Inference Server hosts vLLM and uses LMCache's Valkey RDMA connector to access the external KV-cache service. Valkey server instances run on the CXL KV Storage Server, while benchmark load generation is handled by a separate client host.

Components and Software Configurations

This section defines the component requirements and software configurations across all host roles.

Table 1-2 provides information about the CXL KV Storage Server (`cxL-box`). This server hosts the eight-primary Valkey Cluster (which uses Penguin Velocity server software), the server-side RDMA transport module (`valkey-rdma.so`), and CXL Type-3 memory backing the KV-cache working set.

NOTICE

Do not install NVIDIA drivers, CUDA toolkits, vLLM, LMCache, or client benchmarking software on this server.

Table 1-2. CXL KV Storage Server (`cxL-box`) Components

Component / Package	Recommended / Target Version	Function / Placement Notes
Operating system	Ubuntu® 24.04.4 LTS (Noble)	Base Linux OS; kernel 6.8.0-124-generic
Linux modules	`linux-modules-extra`, `linux-firmware`	Kernel drivers for `mlx5_core`, `mlx5_ib`, and `cxl_*`
RDMA user space	`libibverbs1`, `librdmacm1t64`, `ibverbs-providers` (v50.0-2ubuntu0.2)	RDMA verbs and communication manager libraries

Table 1-2. CXL KV Storage Server (`cxl-box`) Components *(Continued)*

Component / Package	Recommended / Target Version	Function / Placement Notes
RDMA utilities	`ibverbs-utils`, `infiniband-diags`, `iproute2`	Fabric diagnostics (`ibv_devinfo`, `ibstatus`, `rdma`)
NUMA tools	`numactl`, `libnuma1` (v2.0.18+)	Process and memory binding across CXL NUMA servers
Penguin Velocity server and client	`penguin-velocity-server` 9.1.4 (`.deb`) `penguin-velocity-client` 9.1.4	Installs `/usr/bin/penguin-velocity-server` and `/usr/lib/valkey-rdma.so`. Installs the command-line client `/usr/bin/penguin-velocity-cli`.
Valkey RDMA module	Shipped by `penguin-velocity-server` 9.1.4	Server-side RDMA module at `/usr/lib/valkey-rdma.so`. No separate RDMA package is required.
Valkey CLI	`valkey-tools` (v7.2.12+)	Administration tool (`/usr/bin/valkey-cli`)
CXL expander memory	~9.4 TiB CXL Type-3 RAM	CXL Type-3 memory is presented as NUMA nodes 2 and 3. The installed capacity and OS-visible memory per CXL NUMA node are specified in the qualified reference hardware configuration. You should verify node sizes on your server before applying the NUMA map.

Table 1-3 provides information about the GPU Inference Server (`gpu-serve`). This server hosts vLLM tensor-parallel inference, LMCache, the Valkey RDMA client plugin, and high-speed RDMA libraries.

NOTICES

1. Do not run Valkey storage server processes or benchmark client runners (`AIPerf`) on this server.
2. Do not install Penguin Velocity server `.deb` packages on this GPU server. The bundled `lmcache-valkey-next` Python® plugin, not the Penguin Velocity data-store server, is a client connector.

Table 1-3. GPU Inference Server (`gpu-serve`) Components

Component / Package	Recommended / Target Version	Function / Placement Notes
Operating system	Ubuntu 24.04 LTS	Base Linux OS; kernel 6.8.0-136-generic
Python environment	CPython 3.12.3 (range: greater than or equal to 3.10; less than 3.14)	Runtime for vLLM, LMCache, and client plugin
NVIDIA driver	`nvidia-driver-595-server` (v595.71.05)	Driver for 8x NVIDIA H100 80GB HBM3 GPUs
NVIDIA Fabric Manager	`nvidia-fabricmanager-595` (v595.71.05)	Required for NVLink / NVSwitch mesh topology
CUDA® runtime	CUDA 12.9 / PyTorch cu129	Provided via PyTorch binary wheels; no system `nvcc` needed
PyTorch	PyTorch 2.10.0+cu129	Base deep-learning framework
vLLM engine	vLLM 0.17.0	High-throughput LLM serving runtime
LMCache	LMCache 0.5.3	KV cache management framework
Valkey plugin	`lmcache-valkey-next`	LMCache back-end connector for Valkey RDMA
C client library	`libvalkey ENABLE_RDMA=1` (v0.5.0, Git SHA 3180cb6f)	C client library built with RDMA support
Native shim	`liblv_shim.so`	Native bridge binding Python runtime to `libvalkey`
RDMA development libraries	`rdma-core`, `libibverbs-dev`, `librdmacm-dev` (v50.0+)	Development headers for compiling native shims
Build toolchain	`build-essential`, `cmake` (greater than or equal to 3.28), `gcc` (greater than or equal to 13.3)	Required for native client build steps
Transformers	5.9.0 (lab override)	Hugging Face library for model configuration and tokenization. Version 5.9.0 is a reference-environment override due to conflicting vLLM and LMCache dependency requirements. See “Provision Model Weights” on page 1-23 for details.

Table 1-3. GPU Inference Server (`gpu-serve`) Components *(Continued)*

Component / Package	Recommended / Target Version	Function / Placement Notes
NIXL	1.2.0	Included in the reference environment but not used for KV-cache transfers in this deployment, which uses `libvalkey` RDMA or GLIDE TCP. See "Install the Selected KV-Cache Transport" on page 1-19 for transport details.
Redis / `valkey-glide-sync`	8.0.0 / 2.4.1	Python client libraries included in the reference environment. `valkey-glide-sync` supports the GLIDE TCP KV-cache transport, while the RDMA configuration uses `libvalkey`. See "Install the Selected KV-Cache Transport" on page 1-19 for transport details.

NOTE

The GPU Inference Server reference configuration uses LMCache 0.5.3 with PyTorch 2.10.0+cu129 (CUDA 12.9). The standard LMCache 0.5.3 wheel previously installed in the Penguin Solutions lab included a compiled `c_ops` extension linked against CUDA 13, which was incompatible with the PyTorch CUDA 12.9 environment. To address this, the installation procedure now specifies the CUDA 12.9-compatible LMCache 0.5.3 wheel, followed by an explicit verification that the compiled `c_ops` extension loads successfully.

Table 1-4 provides information about the AIPerf Load Generator (`gpu-Loadgen`). This server is a dedicated, closed-loop client host for executing synthetic and benchmark workloads against the serving endpoint.

NOTICE

This host does not require any NVIDIA drivers, CUDA, vLLM, LMCache, or Valkey packages.

Table 1-4. AIPerf Load Generator (`gpu-Loadgen`) Components

Component / Package	Recommended / Target Version	Function / Placement Notes
Operating system	Ubuntu 24.04.4 LTS	Base Linux OS; kernel 6.8.0-124-generic
Python environment	CPython 3.12 (Ubuntu 24.04 default; 3.12.3 on the reference system)	Python virtual environment at `$HOME/vllm-kimi-venv`, used by vLLM, LMCache, and associated client libraries
AIPerf	AIPerf 0.5.0	Closed-loop benchmarking client
System utilities	`tmux`, `curl`, `git`, `openssh-client`	Automation and session management tools

Deployment Prerequisites and Network Specifications

This section describes the deployment prerequisites and network specifications for Penguin Solutions MemoryAI KV Cache Server.

The cluster requires two network planes:

- High-speed data plane: InfiniBand or RoCE network interface cards dedicated to high-throughput RDMA transfers
- Management plane: Standard SSH and administrative communication

For portability, specify the CXL data-plane address as `KV_SERVER_HOST` in `site.env` and then use that variable throughout the installation procedures. For example:

```
assign export=KV_SERVER_HOST=  10.210.11.22
```

Table 1-5 shows an example network configuration. For your installation, use site-approved values for the hostnames and IP addresses. For the port allocations, Valkey uses port 6379 by default for a single instance. Each of the eight Valkey primary instances running on the same CXL KV Storage Server requires a unique listening port and is configurable.

Table 1-5. Network Configuration Example

Server Role	Hostname	Data-Plane IP Address	Management IP Address	Port Allocations
CXL KV Storage Server	`cxl1`	10.210.11.22	10.110.11.22	Ports 6379-6386 (data plane)
GPU Inference Server	`gpu-serve1`	10.210.50.3	10.110.50.3	Port 8000 (vLLM API)
AIPerf Load Generator	`gpu-load1`	10.210.50.1	10.110.50.1	Outbound HTTP/gRPC

Installing and Configuring Penguin Solutions MemoryAI KV Cache Server

Install and configure Penguin Solutions MemoryAI KV Cache Server in the following order:

1. Deploy the CXL KV Storage Server.
2. Deploy the GPU Inference Server.
3. Configure and launch Penguin Solutions MemoryAI KV Cache Server.
4. Deploy the AIPerf Load Generator.

NOTICE

Be sure to verify each component before going to the next step.

Deploying the CXL KV Storage Server

This section discusses how to deploy one CXL KV Storage Server that runs an eight-primary Valkey Cluster using Penguin Velocity server software, with KV-cached data stored in CXL memory.

To deploy the CXL KV Storage Server, perform the following steps in order:

1. “Download the Repository and Packages”
2. “Prepare the Ubuntu Environment”
3. “Install Penguin Velocity 9.1.4”
4. “Configure the Network Bind Address and Ports”
5. “Examine Memory Nodes and CPU Ownership”

6. “Start the Eight-Primary Cluster”
7. “Validate the CXL KV Storage Server Deployment”

After deploying the CXL KV Storage Server, be sure to read the following sections:

- “Perform Operational Checks” on page 1-13
- “Troubleshooting the CXL KV Storage Server Deployment” on page 1-15
- “Performing a Planned Shutdown” on page 1-16
- “Modifying a Configuration” on page 1-16
- “Changing the Number of Primaries” on page 1-17

Download the Repository and Packages

Copy the CXL KV Storage Server deployment `.zip` file supplied by Penguin Solutions to the CXL host, extract it, and run the installation commands from the extracted directory. No access to Penguin Solutions’ internal GitLab is required.

```
mkdir -p "$HOME/memoryai"
unzip -q /path/to/cxl-kv-storage-server-cxl-clean.zip -d
   "$HOME/memoryai"
cd "$HOME/memoryai/cxl-kv-storage-server-cxl-clean"
ls -lh packages/*.deb
dpkg-deb -f packages/penguin-velocity-server_9.1.4_amd64.deb
   Package Version
dpkg-deb -f packages/penguin-velocity-client_9.1.4_amd64.deb
   Package Version
```

No separate `.deb` download URL is required, because the two approved Ubuntu packages are in packages and/or appear when LFS objects have been fetched.

NOTE

Do not install the RPMs on Ubuntu.

Prepare the Ubuntu Environment

Prepare the Ubuntu environment as follows:

```
sudo apt-get update -y
sudo apt-get install --no-install-recommends -y \
   iproute2 pciutils procps util-linux \
   libibverbs1 librdmacm1t64 ibverbs-providers ibverbs-utils
     infiniband-diags \
   numactl libnuma1 openssh-server openssh-client
     valkey-tools dmidecode
```

Alternatively, specify the following command if you prefer to specify a repository-managed equivalent of the preceding commands (this allows you to avoid maintaining a separate installation list):

```
sudo ./scripts/install-lab-packages.sh --role cxl-box
```

Optionally, you can specify `--bootstrap-os` for the preceding command. This option, which requests `linux-image-generic`, `linux-modules-extra-$(uname -r)`, and `linux-firmware`, is not needed for a standard preprovisioned lab host.

Install Penguin Velocity 9.1.4

Install Penguin Velocity 9.1.4 as follows:

```
sudo ./scripts/install-lab-packages.sh --role cxl-box
   --skip-apt --penguin-deb packages
./scripts/install-lab-packages.sh --role cxl-box
   --verify-only
```

The installer validates that both `.deb` files are real packages (not Git LFS pointers), installs the server and client in dependency order, and stops/disables `penguin-velocity-server.service`. That packaged unit otherwise uses `localhost:6379` and conflicts with the eight-primary cluster.

NOTICE

Never enable the packaged single-instance unit alongside this deployment.

Configure the Network Bind Address and Ports

Configure the network bind address and ports as follows:

```
cp site.env.example site.env
$EDITOR site.env
KV_SERVER_HOST="actual_CXL_server_HCA_IP"
VALKEY_SEED_PORT="6379"
VALKEY_PORTS="6379,6380,6381,6382,6383,6384,6385,6386"
```

`KV_SERVER_HOST` must be an IPv4 address assigned to an InfiniBand or RoCE HCA interface on this server; the bring-up script checks interface ownership and `sysfs` HCA association. Do not specify the BMC address, management NIC address, localhost, or another host's IP address. The scripts read the `site.env` file directly. Do not run `apply-site.sh` or manually run `lib-bind.sh`.

Use the following commands to identify the server's network interfaces and verify that the RDMA adapter is recognized before configuring `KV_SERVER_HOST`.

```
ip -4 -o addr show
ls /sys/class/infiniband
ibv_devinfo
```

Examine Memory Nodes and CPU Ownership

Before you copy the reference mapping, examine memory nodes and CPU ownership as follows:

```
numactl --hardware
lscpu -e=CPU,NODE
```

In the reference CXL2 configuration:

- NUMA nodes 2 and 3 provide CXL memory.
- CPU cores 4 through 7 belong to NUMA node 0.
- CPU cores 36 through 39 belong to NUMA node 1.

Verify these assignments against the target server's hardware topology before applying the reference configuration:

```
cp cxl/numa.map.example cxl/numa.map
cat cxl/numa.map
```

Start the Eight-Primary Cluster

Start the eight-primary cluster as follows:

```
sudo ./cxl/bring-up.sh --numa-ok --option-b
```

This command is the recommended RDMA-verified startup. If you do not specify `--option-b`, the script still renders configurations loading `/usr/lib/valkey-rdma.so`. However, without specifying `RDMA` in `MODULE LIST`, the startup software generates a warning rather than failing verification. The `--numa-ok` option is an operator acknowledgment; the `bring-up.sh` script independently verifies node and core membership.

The `bring-up.sh` script automatically generates the required configurations, launches the eight NUMA-pinned Penguin Velocity instances, forms the Valkey Cluster if necessary, and verifies its health. You do not need to manually execute any additional scripts. For non-disruptive verification, use the `--verify-only` option, as described in "Validate the CXL KV Storage Server Deployment" on page 1-13.

Validate the CXL KV Storage Server Deployment

After startup, validate the deployment by running the following command:

```
sudo ./cxl/bring-up.sh --verify-only
```

Confirm that all configured ports respond with `PONG`, the cluster reports `cluster_state:ok` and the expected node count, and the RDMA module is available when required.

NOTE

A loaded RDMA module does not itself prove that application transfers are using RDMA; verify end-to-end GPU/LMCache data movement separately.

Perform Operational Checks

Perform the checks listed in Table 1-6 after installation, a planned restart, or whenever the GPU Inference Server reports remote-cache connectivity problems. Run the commands on the CXL KV Storage Server unless otherwise indicated. Substitute the actual `KV_SERVER_HOST` and configured ports.

Table 1-6. Operational Checks

Check	Action or Expected Observation
Connectivity	The configured HCA data-plane IP address is present locally; all eight configured Valkey ports are reachable. Confirm remote connectivity from the GPU Inference Server as well.
Valkey process health	Eight Penguin Velocity primary processes are running. Their CPU affinity and memory placement agree with `cxl/numa.map`.
Cluster formation	`CLUSTER INFO` reports `cluster_state:ok`, `cluster_known_nodes:8`, and the configured primaries answer `PONG`.
RDMA (if selected)	`MODULE LIST` includes RDMA and the network fabric is healthy. Check application-level RDMA traffic separately, as the module presence alone is insufficient.
CXL capacity	The expected CXL NUMA nodes remain online, with sufficient available memory; verify actual process placement.
Memory budgeting	The four primaries on each CXL node share that node's memory. Do not assign each primary the node's full capacity.

Connectivity and Cluster Checks

Check connectivity and the cluster as follows:

```
# From repository root: automated verification without
   restart
sudo ./cxl/bring-up.sh --verify-only

# Replace the IP with your configured KV_SERVER_HOST
valkey-cli -h KV_SERVER_HOST -p 6379 PING
valkey-cli -h KV_SERVER_HOST -p 6379 CLUSTER INFO
valkey-cli -h KV_SERVER_HOST -p 6379 CLUSTER NODES

# Check listeners and Valkey processes locally
sudo ss -lntp | grep -E
   ':(6379|6380|6381|6382|6383|6384|6385|6386)([[:space:]]
   |$)'
pgrep -af "^/usr/bin/penguin-velocity-server "
```

NUMA and RDMA Checks

Check NUMA and RDMA as follows:

```
# NUMA topology and process placement
numactl --hardware
lscpu -e=CPU,NODE
cat cxl/numa.map
# For a known primary PID:
taskset -cp PID
numastat -p PID

# HCA and RDMA module diagnostics
ip -4 -o addr show
ibv_devinfo
valkey-cli -h KV_SERVER_HOST -p 6379 MODULE LIST
```

Troubleshooting the CXL KV Storage Server Deployment

Table 1-7 describes steps you can take to troubleshoot the CXL KV Storage Server deployment.

Table 1-7. Troubleshooting CXL KV Storage Server Deployment

Issue	Action
Installer fails	Confirm the repository `root`, `--role cxl-box`, exact installer flags, `sudo` permissions and packages/ contents.
NUMA validation fails	Inspect `numactl --hardware`, `lscpu -e=CPU,NODE`, and `cxl/numa.map`. Each configured port needs one row; the core must belong to the specified CPU node; the CXL memory node must exist.
Port does not answer `PING`	Check the process list, `ss` listeners, rendered `cxl/conf/penguin-velocity-`*`port`*`.conf`, HCA bind IP, syslog/process startup output, and host firewall. Confirm that the packaged single-instance systemd unit is disabled.
Cluster state not OK	Check all configured ports, cluster bus connectivity, `CLUSTER NODES`, and form-cluster output. Verify the expected node count and avoid reusing stale cluster identities after changing the number of primaries.
RDMA module absent	Check `/usr/lib/valkey-rdma.so`, rendered `loadmodule/rdma-bind` settings, HCA interface and `ibv_devinfo`; review bring-up output. If you had specified `--option-b`, a missing module is a verification failure.
GPU cannot connect	Compare `KV_SERVER_HOST` on the CXL and GPU hosts; inspect routing, ACLs/firewall, exposed ports, and selected TCP/RDMA client transport. Successful CXL-side `PING` is not full end-to-end validation.

The following commands may be helpful during troubleshooting:

```
# Confirm exact package version and installed module
./scripts/install-lab-packages.sh --role cxl-box
   --verify-only
dpkg-query -W 'penguin-velocity-*'
ls -l /usr/lib/valkey-rdma.so

# Check startup-related paths without modifying cluster state
ls -l cxl/conf/ /root/conf-files/
ls -l /var/log/penguin-velocity/
```

Performing a Planned Shutdown

On the dedicated CXL KV Storage Server, coordinate an inference-side maintenance window before stopping the primaries. The eight Penguin Velocity primaries run as independent background processes, not as a `systemd`-managed service. The packaged `systemd` unit remains disabled and inactive by design. Stopping the primaries makes cached KV data unavailable; the template specifies save "", so do not expect persistence after shutdown.

To gracefully stop the eight Penguin Velocity processes:

```
# From repository root on the CXL host; interrupts all
   primaries
source ./cxl/lib-bind.sh
H="$(cxl_bind_host)"

# Request graceful shutdown without saving cached KV data
   for port in "${PORTS[@]}"; do
     echo "Stopping Valkey on port ${port}"
     valkey-cli -h "$H" -p "$port" SHUTDOWN NOSAVE
done

# Check for remaining processes and listeners
pgrep -af '^/usr/bin/penguin-velocity-server( |$)' || true
for port in "${PORTS[@]}"; do
     sudo ss -lntp "( sport = :${port} )"
done
```

If any primaries remain after the graceful shutdown attempt, send `SIGTERM` as a fallback on the dedicated CXL server:

```
sudo pkill -TERM -f '^/usr/bin/penguin-velocity-server( |$)'
```

Modifying a Configuration

For an IP, port or NUMA change, prepare the new `site.env` and/or `cxl/numa.map`, validate HCA ownership with the `ip -4 -o addr show` command and the physical topology with the `numactl --hardware / lscpu` command, and then schedule and perform a controlled restart with the documented `bring-up` command. Changing the host IP address also requires updating GPU-side LMCache settings and network access-control lists.

After updating and validating the configuration files, restart the cluster to apply the changes:

```
# Start/restart after validated configuration changes
sudo ./cxl/bring-up.sh --numa-ok --option-b
```

Once startup completes, run the verification command to confirm that the cluster is healthy:

```
# Non-disruptive verification after startup
sudo ./cxl/bring-up.sh --verify-only
```

Changing the Number of Primaries

CAUTION

Changing the number of primaries is a distinct, potentially destructive cluster reinitialization.

The `README` instructs you to update `VALKEY_PORTS` and `cxl/numa.map` together and remove /`var/lib/penguin-velocity/*/nodes-*.conf` before you run the `bring-up` command. Do this only after the existing cluster is stopped and the data-loss implications are accepted; the cluster stores reusable KV cache rather than persistent application data. **Do not use this cleanup for an ordinary restart.**

After any reconfiguration, repeat the checks in "Perform Operational Checks" on page 1-13, confirm `cluster_state:ok` with the expected node count, and verify end-to-end KV restoration from the GPU Inference Server.

Deploying the GPU Inference Server

Perform the following steps, in order, to deploy the GPU Inference Server:

1. "Confirm the GPU Inference Server Prerequisites" on page 1-18
2. "Extract the Supplied Inference Server `.zip` File" on page 1-18
3. "Install the Ubuntu, NVIDIA, and Pinned Python Packages" on page 1-19
4. "Install the Selected KV-Cache Transport" on page 1-19
5. "Install the CUDA 12.9-Compatible LMCache `c_ops` Extension" on page 1-21
6. "Configure `site.env` and Render the LMCache `.yaml` File" on page 1-22
7. "Provision Model Weights" on page 1-23
8. "Start Inference and Verify Service" on page 1-24
9. "Validate the External KV Path" on page 1-25

After deploying the GPU Inference Server, be sure to read "Troubleshooting the GPU Inference Server Deployment" on page 1-26.

Confirm the GPU Inference Server Prerequisites

Confirm that the following prerequisites for deploying the GPU Inference Server have been met:

- ❏ Ubuntu 24.04 x86-64 GPU server; eight H100 GPUs for reference `GPU_PROFILE=tp8`, working NVIDIA fabric, and root privileges to install driver packages.
- ❏ Access to public Ubuntu, PyPI, PyTorch CUDA wheel index, GitHub (LMCache and `valkey-io/libvalkey`), and Hugging Face. Approved enterprise mirrors are acceptable if they contain the same artifacts.
- ❏ CXL KV Storage Server is configured and reachable by its data-plane HCA address. For RDMA, the server must expose RDMA-enabled Valkey, and the network fabric must be configured.
- ❏ At least 150 GiB free for the gated-model checkpoint, appropriate Hugging Face access and licensing, and reboot window after NVIDIA driver installation.

Extract the Supplied Inference Server `.zip` File

Copy the supplied `.zip` file to the GPU Inference Server as the account that will run inference. The following commands use the exact archive name delivered for this review. Replace the source path if the filename differs.

```
mkdir -p "$HOME/memoryai"
unzip -q /path/to/inference-server-inference-clean.zip -d
   "$HOME/memoryai"
cd "$HOME/memoryai/inference-server-inference-clean"
ls -l README.md scripts/install-lab-packages.sh \
   operations/lab-pins.env plugin/lmcache-valkey-next/
```

NOTE

All subsequent relative commands are run from this extracted directory unless stated otherwise. No private GitLab access or Git credentials are needed.

Install the Ubuntu, NVIDIA, and Pinned Python Packages

Preserve the repository's installation sequence and pinned versions (see Table 1-3 for version information). The installer enables Ubuntu universe/multiverse for `nvidia-driver-595-server`, installs required APT packages, creates `$HOME/vllm-kimi-venv`, and then installs vLLM/PyTorch and LMCache-related Python packages in two `pip` phases. Do not merge those phases: vLLM and LMCache have conflicting declared transformers ranges; the recorded lab configuration resolves them by installing transformers 5.9.0 in phase two.

```
# On the GPU server, from the extracted ZIP root
sudo -E ./scripts/install-lab-packages.sh --role gpu-serve
   --nvidia
sudo reboot
# After reboot, return to the ZIP root
cd "$HOME/memoryai/inference-server-inference-clean"
./scripts/install-lab-packages.sh --role gpu-serve
   --verify-only
nvidia-smi
nvidia-smi -L
```

NOTICE

Do not proceed if the driver, package version checks, or the expected number of GPUs fail. The original `--verify-only` command checks software pins; the compiled `c_ops` check described in "Install the CUDA 12.9-Compatible LMCache `c_ops` Extension" on page 1-21 is an additional mandatory gate.

Install the Selected KV-Cache Transport

Install the KV-cache transport using **one** of the following methods:

NOTICE

You must use the same transport method when you perform the steps in "Configure `site.env` and Render the LMCache `.yaml` File" on page 1-22

- **RDMA using libvalkey and lmcache-valkey-next**

  For RDMA, use the following unmodified repository installer. It clones the public `libvalkey` repository, checks out the pinned commit, builds with `ENABLE_RDMA=1`, compiles `liblv_shim.so`, installs the plugin already included in the `.zip` file, and runs the connector preflight.

```
# Run as the normal inference account, NOT through sudo
./scripts/install-lab-packages.sh --role gpu-serve
   --option-b
# Confirm the installed native library and Python connector
export LIBVALKEY_PREFIX="$HOME/opt/libvalkey"
export
   LD_LIBRARY_PATH="$LIBVALKEY_PREFIX/lib:${LD_LIBRARY_
   PATH:-}"

"$HOME/vllm-kimi-venv/bin/python" -m
lmcache_valkey_next.check_rdma
```

  NOTES

  1. The source of the preceding is public `https://github.com/valkey-io/libvalkey.git`, pinned to commit 3180cb6f7a7acf5224de13ae5deb44ff5744d92b (libvalkey 0.5.0). The plugin source comes from the supplied `.zip` file, not from a separate customer clone.
  2. The `--option-b` command also reruns the pinned pip installation in its default mode. Complete it before you perform the steps in "Install the CUDA 12.9-Compatible LMCache `c_ops` Extension" on page 1-21, or execute those steps afterward.

- **TCP using the built-in GLIDE client**

  No extra connector build is needed. The base installation already provides `valkey-glide-sync==2.4.1`. Set `KV_TRANSPORT=tcp` in `site.env`. Do not run `--option-b` for a TCP-only installation.

Install the CUDA 12.9-Compatible LMCache `c_ops` Extension

Perform the following commands:

> NOTE
>
> This step is required for both TCP and RDMA installations.

```
# Download the CPython 3.12 / x86-64 CUDA 12.9 wheel from
   GitHub
VENV="$HOME/vllm-kimi-venv"
WHEEL="lmcache-0.5.3-cp312-cp312-manylinux_2_27_x86_64.
   manylinux_2_28_x86_64.whl"
BASE="https://github.com/LMCache/LMCache/releases/download/
   v0.5.3-cu129"
mkdir -p "$HOME/memoryai/downloads"
cd "$HOME/memoryai/downloads"
curl -fL --retry 3 -o "$WHEEL" "$BASE/$WHEEL"

# Official SHA-256 from the LMCache release assets
printf '%s  %s\n' \
'3587d26a23e942b774589c88ea4cfb019af53474aba84dc772dae5900f9
   ad2cb' \
"$WHEEL" | sha256sum -c -

# Replace only LMCache; do not change torch or any
   dependencies
"$VENV/bin/python" -m pip install --force-reinstall --no-deps
   "./$WHEEL"
```

Do not use the standard PyPI `lmcache==0.5.3` wheel again afterward; it may restore the incompatible native extension. If you repeat the repository pip installer, you need to repeat the following steps:

```
# Mandatory CUDA and compiled-extension verification
"$HOME/vllm-kimi-venv/bin/python" - <<'PYCODE'
import sys, importlib.metadata as m, torch
import lmcache.c_ops as c
assert sys.version_info[:2] == (3, 12)
assert m.version('lmcache') == '0.5.3'
```

(Continued on next page)

```
assert torch.__version__ == '2.10.0+cu129' and
   torch.version.cuda == '12.9'
assert torch.cuda.is_available()
assert hasattr(c, 'multi_layer_kv_transfer')
print('PASS: CUDA 12.9 compiled c_ops loaded:', c.__file__)
PYCODE
```

NOTICE

Do not accept fallback. If this import fails or the serving logs report the message `c_ops compiled extension not found`, stop and correct the installation. An HTTP health response is not proof that compiled KV transfers are enabled. Actual external KV-cache reads and performance still require end-to-end verification.

Configure `site.env` and Render the LMCache `.yaml` File

Configure the GPU Inference Server `site.env` file as follows:

```
cd "$HOME/memoryai/inference-server-inference-clean"
cp site.env.example site.env
${EDITOR:-vi} site.env
```

Table 1-8 lists the settings that you configure in the `site.env` file.

Table 1-8. GPU Inference Server `site.env` Configuration Settings

Setting	Configuration
`KV_TRANSPORT`	Either `tcp` or `rdma` (whichever you specified in "Install the Selected KV-Cache Transport" on page 1-19)
`KV_SERVER_HOST`	CXL server data-plane HCA IP address (not management/BMC)
`SERVE_HOST` / `SERVE_PORT`	GPU host serving IP / default 8000
`VALKEY_SEED_PORT` / `VALKEY_PORTS`	Reference 6379 / 6379–6386
`MODEL`	nvidia/Llama-3.1-Nemotron-70B-Instruct-HF
`GPU_PROFILE`	`tp8` for the eight-GPU reference
`CLIENT_SSH` / `CLIENT_OVERFLOW`	Required only for optional remote AIPerf lab smoke test

After you edit `site.env`, run the following command to generate the LMCache `.yaml` file for the selected transport. If `KV_TRANSPORT` changes, rerun the script to generate the new transport configuration. The previously generated transport `.yaml` file is not automatically refreshed.

```
./scripts/render-serve-config.sh
# TCP:  server/penguin-velocity.yaml
# RDMA: examples/lmcache-d96-valkey-next-rdma.yaml
```

Table 1-9 compares selected settings in the generated TCP (GLIDE) and RDMA (`valkey-next`) LMCache configurations. The complete `.yaml` file for the chosen transport is generated automatically.

Table 1-9. Key LMCache Configuration Settings for TCP and RDMA

TCP (GLIDE) `.yaml`	RDMA (`valkey-next`) `.yaml`
`server/penguin-velocity.yaml`	`examples/lmcache-d96-valkey-next-rdma.yaml`
`remote_url: valkey://`*`KV_SERVER_HOST`*`:6379`	`remote_storage_plugins: ["valkey-next"]`
`valkey_mode: cluster`	`valkey-next.transport: rdma`
`valkey_num_workers: 32`	`valkey-next.cluster: true`
`chunk_size: 1024`	`chunk_size: 1024`

The TCP file uses the built-in GLIDE client and a single seed to discover cluster nodes. The RDMA file loads the bundled plugin and configures RDMA transport and multi-primary discovery. Both set `local_cpu: false` and `max_local_cpu_size: 50`. Inspect the generated file and explicitly set `LMCACHE_CONFIG_FILE` in the server environment; otherwise the launcher may default to the TCP `.yaml` file.

Provision Model Weights

The model checkpoint is not inside the `.zip` file. Obtain access to `nvidia/Llama-3.1-Nemotron-70B-Instruct-HF` on Hugging Face, accept the NVIDIA/Llama license, and provide approximately 132 GiB for the checkpoint (at least 150 GiB free recommended). Use the same `HF_HOME` during download and serving.

```
export VLLM_VENV="$HOME/vllm-kimi-venv"
export HF_HOME="${HF_HOME:-$HOME/.cache/huggingface}"
"$VLLM_VENV/bin/huggingface-cli" login
```

(Continued on next page)

```
"$VLLM_VENV/bin/python" -c  "from huggingface_hub import
   snapshot_download;
print(snapshot_download('nvidia/Llama-3.1-Nemotron-70B-
   Instruct-HF'))"
```

If the installed `huggingface_hub` package provides hf auth login instead of `huggingface-cli` login, use the available supported CLI for the installed version. Keep `MODEL` set to the model hub identifier unless a local model path is deliberately provisioned and validated.

Start Inference and Verify Service

Run the server as the ordinary inference user. The launcher is foreground-oriented and not a packaged customer `systemd` unit. Run it in a managed terminal or approved process supervisor.

The following example includes a selection for both transports. The following commands automatically select the correct LMCache configuration based on `KV_TRANSPORT`. Run the entire command block without modifying its transport-selection logic.

```
cd "$HOME/memoryai/inference-server-inference-clean"
set -a; source ./site.env; set +a
export VLLM_VENV="$HOME/vllm-kimi-venv"
export PATH="$VLLM_VENV/bin:$PATH"
export MEMORYAI_ROOT="$PWD"
export KV_VRAM_PROFILE=spill
if [[ "$KV_TRANSPORT" == rdma ]]; then
   export
     LMCACHE_CONFIG_FILE="$PWD/examples/lmcache-d96-valkey-
     next-rdma.yaml"
   export LIBVALKEY_PREFIX="$HOME/opt/libvalkey"
   export
     LD_LIBRARY_PATH="$LIBVALKEY_PREFIX/lib:${LD_LIBRARY_
     PATH:-}"
   export LMCACHE_VALKEY_NEXT_DISABLE_HASH_TAG=1
else
   export
     LMCACHE_CONFIG_FILE="$PWD/server/penguin-velocity.yaml"
fi
[[ -s "$LMCACHE_CONFIG_FILE" ]] || { echo 'LMCache YAML
   missing'; exit 1; }
./overflow/scripts/server/start-lmcache.sh
```

In a second terminal, validate HTTP health and the external Valkey Cluster. Set the address according to the configured *SERVE_HOST*; 127.0.0.1 works for a local health check only if the service listens on a suitable interface.

```
curl -fsS "http://SERVE_HOST:8000/health"
curl -fsS "http://SERVE_HOST:8000/v1/models"
valkey-cli -h KV_SERVER_HOST -p 6379 PING
valkey-cli -h KV_SERVER_HOST -p 6379 CLUSTER INFO
```

Confirm the server log uses the intended `.yaml` file. For RDMA, look for `lmcache-valkey-next loaded` and `transport=rdma`; the connector preflight must already pass. In all cases, reject the message `c_ops compiled extension not found` or a CUDA native import failure.

Validate the External KV Path

HTTP health, cluster `PING`, and a successful plugin import do not establish remote KV-cache restoration. Before you accept the deployment, perform the following tasks, in order. , to validate the actual external KV path:

1. On a dedicated validation system, send a prompt containing a repeated prefix longer than the configured 1024-token LMCache chunk size and confirm external cache writes with LMCache/Valkey logs or counters.
2. Stop and restart only the inference-side vLLM process. Keep the CXL KV Storage Server running and its cached keys intact during this time.

   NOTICE

   Do not flush or restart the remote Valkey Cluster between the first and second inference requests. Doing so would invalidate the restore test.

3. Reissue the same test prompt used in step 1. Because vLLM has restarted while the external cache remains intact, the second request can be checked for retrieval of previously stored KV-cache data rather than local GPU prefix reuse.
4. For RDMA, confirm RDMA transport in the connector logs and validate actual fabric traffic. The presence of a module is not, by itself, proof.
5. Record TTFT and, for performance acceptance, effective host-to-GPU copy behavior. The compatible `c_ops` import is necessary but is not, by itself, a benchmark result.

Optionally, you can also perform a lab smoke test. The `README` file includes an AIPerf-based automated smoke test that requires a generated RAG dataset and remote load generator. It starts/stops vLLM, and the sweep can flush Valkey cache. These workflows are optional, disruptive, and not routine customer service-health commands.

Troubleshooting the GPU Inference Server Deployment

Table 1-10 describes actions you can take to troubleshoot the GPU Inference Server deployment.

Table 1-10. Troubleshooting GPU Inference Server Deployment

Issue	Action
Software installer fails	Verify current directory, public APT/PyPI availability, rebooted 595.71.05 driver, required Python 3.12 and script execution permissions.
CUDA `c_ops` missing/slow fallback	Perform the actions described in "Install the CUDA 12.9-Compatible LMCache `c_ops` Extension" on page 1-21 after the final `pip/--option-b` installation; confirm SHA-256 and Python extension import; reject fallback.
RDMA plugin fails	Check `--option-b` public libvalkey checkout/build, local shim file, `LD_LIBRARY_PATH`, and `check_rdma`; validate CXL RDMA module and network.
vLLM uses incorrect back-end	Check `KV_TRANSPORT`, rerun `render-serve-config.sh`, inspect `LMCACHE_CONFIG_FILE` and serving log.
CXL store unreachable	Verify data-plane CXL host IP, port 6379, `CLUSTER INFO`, routing, ACLs, cluster mode and matching selected transport.
Model fails to load	Validate gated license/access, `HF_HOME`, available storage, `nvidia-smi`, `GPU_PROFILE`, and model identifier.

Configuring and Launching Penguin Solutions MemoryAI KV Cache Server

Perform the following steps, in order, to configure and launch Penguin Solutions MemoryAI KV Cache Server:

- "Configure the LMCache RDMA `.yaml` file" on page 1-27
- "Launch the vLLM Inference Engine" on page 1-27

Configure the LMCache RDMA `.yaml` file

Create an LMCache configuration file named `lmcache-valkey-rdma.yaml` as follows:

```
# LMCache Valkey RDMA Configuration
chunk_size: 256
local_device: "cuda"
remote_backend: "lmcache_valkey_next.ValkeyRemoteBackend"
valkey-next.host: KV_SERVER_HOST
valkey-next.port: 6379
valkey-next.transport: "rdma"
valkey-next.cluster: true
remote_serde: "naive"
save_chunk_meta: false
```

Launch the vLLM Inference Engine

Launch the vLLM inference engine by starting the vLLM server with Tensor Parallelism set to 8 (this tells vLLM to use all eight GPUs), using your target LLM model weights:

```
export LIBVALKEY_PREFIX=$HOME/opt/libvalkey
export
LD_LIBRARY_PATH=$LIBVALKEY_PREFIX/lib:${LD_LIBRARY_PATH:-}
export LMCACHE_CONFIG_FILE=lmcache-valkey-rdma.yaml

python -m vllm.entrypoints.openai.api_server \
  --model /path/to/model-weights \
  --tensor-parallel-size 8 \
  --port 8000
```

Deploying the AIPerf Load Generator

The AIPerf Load Generator is a dedicated Ubuntu HTTP client for generating reproducible request workloads against the GPU Inference Server. It does not run inference or manage the external KV cache.

The AIPerf Load Generator requires no NVIDIA drivers, CUDA, Fabric Manager, vLLM, LMCache, Penguin Velocity server, Valkey, or `libvalkey`. It connects to the GPU Inference Server via HTTP. The cache transport between the GPU and CXL servers is configured independently.

Perform the following steps, in order, to deploy the AIPerf Load Generator:

1. “Confirm the AIPerf Load Generator Prerequisites” on page 1-28
2. “Extract the `.zip` File” on page 1-29
3. “Install the AIPerf and Ubuntu Prerequisites” on page 1-29

4. “Configure the GPU Inference Server Endpoint” on page 1-30
5. “Verify the Load Generator and API” on page 1-31
6. “Run a One-Request Smoke Benchmark” on page 1-32
7. “Run a Measured Workload” on page 1-33
8. “Locate and Review Benchmark Results” on page 1-33

After deploying the AIPerf Load Generator, be sure to read “Troubleshooting the AIPerf Load Generator Deployment” on page 1-34.

Confirm the AIPerf Load Generator Prerequisites

Confirm that the following prerequisites for deploying the AIPerf Load Generator have been met:

- ❏ Operating system: Ubuntu 24.04 x86-64 with Python 3.12 (the verification tool requires Python 3.12.x).
- ❏ User permissions: Normal user permissions for the scripts; `sudo` permissions only if the installer must install missing Ubuntu packages.
- ❏ Software access: PyPI over HTTPS, or an enterprise mirror providing the pinned public packages.
- ❏ Network: TCP reachability to the GPU Inference Server vLLM port (reference 8000). No direct CXL KV Storage Server access is required.
- ❏ GPU Inference Server: Complete the GPU Inference Server startup and verification procedure before installing or testing the AIPerf Load Generator. Its verification script checks model availability and submits a short inference request.
- ❏ Input datasets: Synthetic prompts do not need a file. Custom JSONL datasets must be provided separately on the load-generator host.

The installer checks for the following Ubuntu packages and installs only those that are missing: `python3`, `python3-venv`, `python3-pip`, `tmux`, `openssh-client`, `git`, `curl`, `wget`, `rsync`, and `ca-certificates`. The installer creates `$HOME/aiperf-venv` and installs `aiperf==0.5.0` and `pyarrow==25.0.0`. The PyArrow pin matches the working `gpu1` environment with NumPy 1.26.4.

Extract the `.zip` File

Use the following commands to extract the supplied `.zip` file and copy it to the AIPerf Load Generator.

NOTES

1. The following example uses the actual `.zip` file and top-level directory name from the supplied package. If your `.zip` file has a different name, substitute its location.
2. The working directory must contain the `README.md` file and `load-gen` directory.
3. You do not require access to Penguin Solutions' private GitLab repository.

```
command -v unzip >/dev/null || { sudo apt-get update && sudo
   apt-get install -y unzip; }
mkdir -p "$HOME/memoryai"
unzip -q /path/to/client-load-generator-load-gen.zip -d
   "$HOME/memoryai"
cd "$HOME/memoryai/client-load-generator"
chmod +x load-gen/*.sh
ls -l README.md load-gen/*.sh load-gen/load-gen.env.example
```

Install the AIPerf and Ubuntu Prerequisites

Run the following command from the extracted package root to install the AIPerf and Ubuntu prerequisites:

```
./load-gen/install-load-gen.sh
```

The installer handles missing APT packages, creates a dedicated Python virtual environment, installs the pinned packages from the public Python index, and checks the installed AIPerf version. You can re-run the installer if necessary.

NOTICE

Do not install Python dependencies using a system-wide `sudo pip` command.

If the installation succeeded, the following output appears:

```
PASS: aiperf 0.5.0 is installed in /home/user/aiperf-venv
```

Configure the GPU Inference Server Endpoint

Configure the GPU Inference Server endpoint as follows:

1. Run the following command to find the model ID value returned by the running GPU Inference Server, and make a note of this value. Replace *GPU_INFERENCE_SERVER_IP* with the actual server IP address, and use the model ID returned by `/v1/models` when setting `VLLM_MODEL`.

   ```
   curl http://GPU_INFERENCE_SERVER_IP:8000/v1/models
   ```

2. Run the following commands to create a copy of the example `load-gen.env` file and to open the new file for editing.

   NOTE

   Do not edit the actual `load-gen.env.example` file.

   ```
   cp load-gen/load-gen.env.example load-gen/load-gen.env
   vi load-gen/load-gen.env
   ```

3. Edit your copy of the `load-gen.env` file to configure the GPU Inference Server endpoint and workload parameters (that is, the hostname or IP address, listening port, and served model ID). The load-generator scripts read this file automatically; do not modify `load-gen.env.example`.

   ```
   VLLM_HOST="REPLACE_WITH_GPU_INFERENCE_SERVER_IP_OR_HOSTNAME"
   VLLM_PORT=8000
   VLLM_MODEL="nvidia/Llama-3.1-Nemotron-70B-Instruct-HF"
   ```

Table 1-11 lists the workload settings you can modify in the `load-gen.env` file.

Table 1-11. Workload Settings in the `load-gen.env` File

Parameter	Description
`AIPERF_ENDPOINT_TYPE`	`chat`: Uses `/v1/chat/completions`. `completions`: Also supported.
`AIPERF_STREAMING`	true: Streaming responses for TTFT and inter-token latency measurements.
`AIPERF_CONCURRENCY`	32: Simultaneous in-flight requests.
`AIPERF_REQUEST_COUNT`	200: Total requests for a run.
`AIPERF_TIMEOUT`	600:Per-request time-out, in seconds.

Table 1-11. Workload Settings in the `load-gen.env` File *(Continued)*

Parameter	Description
`AIPERF_INPUT_FILE`	Empty: Synthetic prompts. Non-empty: Path to a JSONL file on this host.
`AIPERF_DATASET_TYPE` / `AIPERF_DATASET_SAMPLING`	`multi_turn` / `sequential`: Used with an input file.
`AIPERF_INPUT_TOKENS_MEAN` / `STDDEV`	550 / 0: For synthetic prompts only.
`AIPERF_OUTPUT_TOKENS_MEAN` / `STDDEV`	Empty / 0: Omit output cap by default. Set as needed.
`AIPERF_USE_LEGACY_MAX_TOKENS`	0: Default value. 1: Sends `max_tokens` for servers requiring the legacy field.
`AIPERF_WARMUP_REQUEST_COUNT`	Empty: By default (no separate warm-up).
`AIPERF_TOKENIZER`	Empty: Uses `VLLM_MODEL`. You can specify a different tokenizer ID.
`RESULTS_DIR`	Directory where benchmark results are stored. Defaults to `$HOME/aiperf-results`; may be changed to another writable location.

Verify the Load Generator and API

NOTICE

Before continuing, confirm that the GPU Inference Server responds on its API endpoint and lists the expected model. If not, perform the tasks in "Start Inference and Verify Service" on page 1-24 and correct the failure before proceeding.

Run the verification script:

```
./load-gen/verify-load-gen.sh
```

The script checks Python 3.12 and AIPerf 0.5.0, DNS/IP resolution, TCP connectivity, `GET /v1/models`, exact served-model ID matching, and a short `POST /v1/chat/completions` request.

If the verification succeeds, the following output appears:

```
PASS: AIPerf load-generator environment is ready.
PASS: vLLM endpoint is reachable.
PASS: model is available.
PASS: inference smoke request succeeded.
```

NOTES

1. If verification fails, review the `FAIL` message, correct the reported issue, and rerun `./load-gen/verify-load-gen.sh`. See Table 1-12 for common corrective actions.
2. A successful verification confirms the HTTP request path. To verify that the GPU Inference Server is using the CXL KV Storage Server, see "Validate the External KV Path" on page 1-25.

Run a One-Request Smoke Benchmark

For the initial smoke test, temporarily set the following values in the `load-gen.env` file:

```
AIPERF_CONCURRENCY=1
AIPERF_REQUEST_COUNT=1
AIPERF_INPUT_FILE=
AIPERF_INPUT_TOKENS_MEAN=32
AIPERF_INPUT_TOKENS_STDDEV=0
AIPERF_OUTPUT_TOKENS_MEAN=8
AIPERF_OUTPUT_TOKENS_STDDEV=0
AIPERF_USE_LEGACY_MAX_TOKENS=1
```

Run the following command to execute the AIPerf smoke benchmark:

```
./load-gen/run-aiperf.sh
```

The script prints the exact `aiperf` profile command, runs it, and stores artifacts. Confirm that the exported metrics show one completed request and zero errors.

After the smoke test succeeds, update the `load-gen.env` values that you temporarily set above to match the intended benchmark workload.

Run a Measured Workload

To run a measured workload:

1. For a representative multi-turn test, copy the dataset to the AIPerf Load Generator separately and point `AIPERF_INPUT_FILE` to the absolute path. The supplied `.zip` file does not include any JSONL workload data.

   An example of `load-gen.env` settings for a dataset-backed run follow:

   ```
   AIPERF_INPUT_FILE=/path/to/customer_workload.jsonl
   AIPERF_DATASET_TYPE=multi_turn
   AIPERF_DATASET_SAMPLING=sequential
   AIPERF_STREAMING=true
   AIPERF_CONCURRENCY=32
   AIPERF_REQUEST_COUNT=2500
   ```

2. Execute the selected workload by starting the benchmark from the package root as follows:

   ```
   ./load-gen/run-aiperf.sh
   ```

Locate and Review Benchmark Results

You can find the results of the AIPerf Load Generator in a unique subdirectory of `RESULTS_DIR`. An example directory structure follows:

```
$HOME/aiperf-results/20261019T173000-c32-n200/
   command.txt
   environment.txt
   aiperf-output.log
   profile_export_aiperf.json
   profile_export_aiperf.csv
   profile_export.jsonl
   logs/aiperf.log
```

In the preceding example, the results directory (`aiperf-results`) includes the run timestamp showing the local date (`20261019`) and time (`T173000`), concurrency level (`c32`), and request count (`n200`). The timestamp uses the load-generator host's local clock unless otherwise configured.

Review `profile_export_aiperf.json` or the CSV export for completed requests, errors, latency, and throughput. For streaming workloads, inspect TTFT and inter-token latency. Use `command.txt` and `environment.txt` to confirm the exact test configuration before comparing runs.

Troubleshooting the AIPerf Load Generator Deployment

Table 1-12 describes actions you can take to troubleshoot the AIPerf Load Generator deployment.

Table 1-12. Troubleshooting the AIPerf Load Generator Deployment

Issue	Recommended Check
Missing `load-gen.env` / empty `VLLM_HOST`	Copy the example, and then set the GPU Inference Server address, port, and model ID.
AIPerf executable or wrong Python version	Re-run `install-load-gen.sh`. Use Ubuntu 24.04 with Python 3.12.
AIPerf startup / PyArrow import failure	Verify `aiperf==0.5.0` and `pyarrow==25.0.0` in `$HOME/aiperf-venv`, and then re-run the installer.
DNS/IP or TCP connection failure	Check `VLLM_HOST`, `VLLM_PORT`, GPU server status, routing, and host firewall.
`GET /v1/models` fails / model ID not listed	Confirm that the vLLM API is running. Use the exact ID returned from `/v1/models`.
Chat smoke request fails	Inspect HTTP status and response body printed by `verify-load-gen.sh`. Check API/model readiness.
JSONL input missing or empty	Copy a valid dataset onto the load generator and set `AIPERF_INPUT_FILE` to an existing path.
Scored run returns errors or times out	Inspect `aiperf-output.log` and `logs/aiperf.log`. Assess time-out, concurrency, and GPU Inference Server load.

Verifying and Troubleshooting

This section describes the following topics:

- “Verifying Server Health” on page 1-34
- “Reverting from RDMA to GLIDE TCP Mode” on page 1-35
- “Enforcing LRU Memory Eviction” on page 1-35

Verifying Server Health

Table 1-13 lists the system components to be verified, the commands or inspections used to verify them, and the expected results for a healthy component. **If any verification check fails, do not proceed with performance testing.** Troubleshoot the affected component using the corresponding deployment and troubleshooting instructions, correct the issue, and then rerun the verification.

Table 1-13. Verifying Server Health

Failed Check	Where to Investigate First
CXL storage cluster	Eight server processes, configured ports, `CLUSTER INFO`, and `CLUSTER NODES`.
Valkey RDMA module	`/usr/lib/valkey-rdma.so`, `MODULE LIST`, HCA, and fabric status.
GPU mesh / Fabric Manager	`nvidia-smi`, GPU count, and Fabric Manager unit/logs.
Client RDMA preflight	`libvalkey` installation, native shim, library search path, and connector import.
vLLM startup / remote cache	Serving logs, LMCache `.yaml` and connector selection, model, and remote store reachability.

Reverting from RDMA to GLIDE TCP Mode

To revert the GPU Inference Server from RDMA to GLIDE TCP, run the following commands:

```
# From the extracted GPU inference-server directory
# First edit site.env and set KV_TRANSPORT=tcp
./scripts/render-serve-config.sh
export
   LMCACHE_CONFIG_FILE="$PWD/server/penguin-velocity.yaml"
# Restart vLLM using the documented Start Inference procedure.
```

The preceding commands set `KV_TRANSPORT=tcp` in the `site.env` file, regenerate the LMCache configuration, select the generated TCP `.yaml` file, and restart the inference process. Verify API readiness and external cache connectivity before resuming the workload.

Enforcing LRU Memory Eviction

The reference deployment does not configure an explicit per-primary `maxmemory` limit. Four Valkey primaries share each CXL memory node.