Enabling AI Storage Benchmark Evolution at the Pace of AI

April, 2026

Wes Vaske
SMTS, Storage Solutions Architect

micron Intelligence Accelerated™

SDC | StorageAI™

Agenda

- Introduction
- MLPerf Storage v3.0
- SNIA AI Data Workloads TWG
- Workloads
  - Checkpointing Workload
  - VectorDB
  - KV Cache

Wes Vaske

Micron: Technical Staff, Storage Solutions Architect

SNIA: Chair of AI Data Workloads TWG

- 15 years in Performance Engineering
  - Databases
  - Storage Arrays
  - Software Defined Storage
  - AI & Storage

- 27 years running benchmarks
  - Back when a lead pencil was a primary overclocking tool

- How do benchmarks get used?
  - Product development
  - Validation testing
  - Sizing hardware requirements
  - Purchasing decisions
  - Performance leaderboards
    - “trash talk with rules”

MLPerf Storage – A brief history

Benchmarking AI for storage has 2 big hurdles:

1. **AI systems requires costly accelerators**
   - $30k => $50k => $100k+?
2. **The available datasets are small compared to datasets used in the industry**
   - Example: Recommendation Systems
     - Criteo dataset used for AI training benchmarks is 1TB pre-processed and 300GB post-processed
     - Meta discusses recommendation training jobs reading 25+ Petabytes of semi-processed data[1]
       - In 2021
     - 100x different

- MLCommons has benchmarks for many parts of the AI pipeline but they remove storage dependencies to focus on compute
- MLPerf Storage was developed to address the gap
  - Develops benchmark tools that can closely emulate real AI compute on traditional CPU-based servers
  - Generates large artificial datasets with similar characteristics to the real datasets
    - Distribution of file sizes, size of samples, number of samples per file, etc
- MLPerf Storage has published 3 sets of results
  - A version is a definition of how to run the benchmarks in a way that can be compared across vendors / solutions with specific workload configurations
  - v0.5 and v1.0 were training ingest only
  - v2.0 added checkpointing of large models

1. Understanding Data Storage and Ingestion for Large-Scale Deep Recommendation Model Training

MLPerf Storage – v3.0 Enhancements

New Workloads

- Vector Database
- KV Cache Management

Updated Models for Training Ingest

- DLRM (recommendation)
- Retinanet (image classification)
- Flux (text to image)

Updated Data Loaders

- S3 Support
- Sparsely accessed parquet for DLRM

Ease of use & documentation updates

- Better package management
- Dependency version locking
- Improved cluster-wide metric collection
- Better validation with fast fail
- Re-architected for better modularity to support increased pace of new benchmarks
- Code tests!

New SNIA TWG: AI Data Workloads TWG

Chair: Wes Vaske (Micron). Vice-Chair: Glyn Bowden (HPE)

Chairs:

- Wes Vaske (Micron)
- Glyn Bowden (HPE)

Have opinions you want to share?

6:15pm - Birds of a Feather: Community discussion of Storage for AI Workload Definitions, Characterization, and Benchmarking needs and wants

Aim to address gaps in the benchmarking ecosystem of storage for AI

- Lack of standard definition for test and validation
- Excessive overhead in existing tools
- Scaling parameters to represent future systems

The AI Data Workload TWG defines and develops synthetic workload definitions of an isolated storage target (e.g., a single SSD, NAS, remote storage device) with respect to how the entire AI pipeline interacts with the storage target.

This TWG produces technical white papers on how to recreate each workload generated and produces reference SNIA Software to generate the defined workload patterns.

The technical whitepapers are publicly available through SNIA's technical process and SNIA Software is distributed through the 3-Clause BSD License – Open Source Initiative.

Program of Work: In the next 6 months, deliver whitepapers defining workloads and the associated reference implementation and execution methodology.

- KV Caching
- Training Checkpointing
- Training Ingest

Following 6-months:

- Accelerator Initiated GNN Training
- Vector DB Indexing & Search
- AI Model Offloading (non-LLM)

Checkpointing

Checkpoints are Big

- Llama 8B is ~16GB in FP16
  - *Writing or reading 16GB in a data center is trivial!*

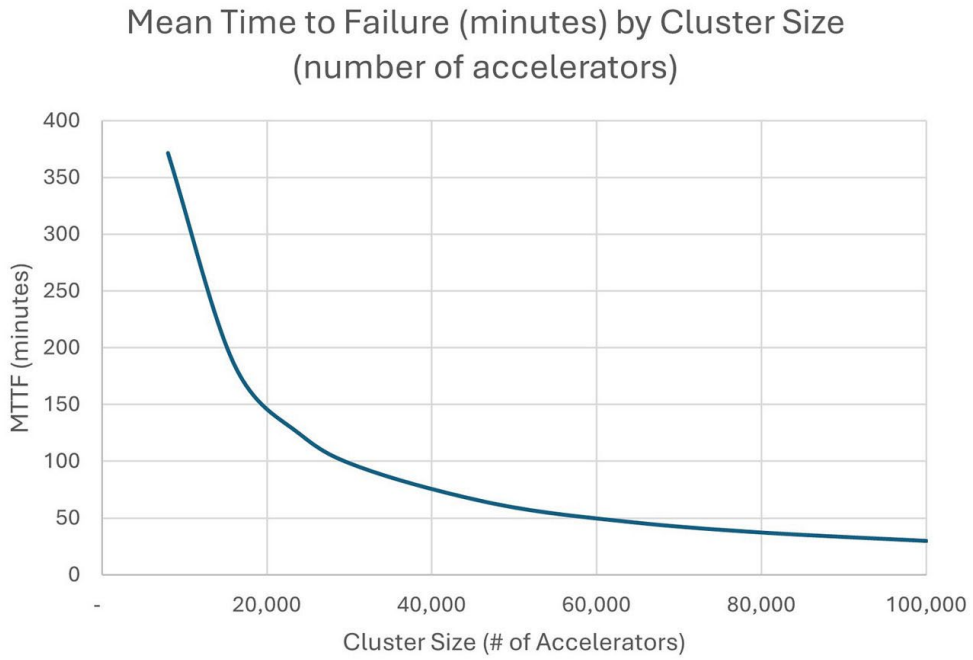
- Checkpoints are more than just the model weights
  - Optimizer state is quite large
    - 6 – 12 Bytes per parameter
    - 8 billion parameters = 48 – 96 GB
    - 405B parameters = 2.4 – 4.8 TB
    - 1T parameters = 6 – 12 TB

4:40 – 5:10pm: *Take a Break: A Deep Dive into Checkpointing*

John Mazzie | Micron Technology - Member of Technical Staff, Systems Performance Engineer

Clusters will fail

- Failure rates of the cluster relate to the power of failure domains
  - (failure rate of GPU)^(# GPUs)
- Large scale clusters (>100k GPUs) face high frequency failures

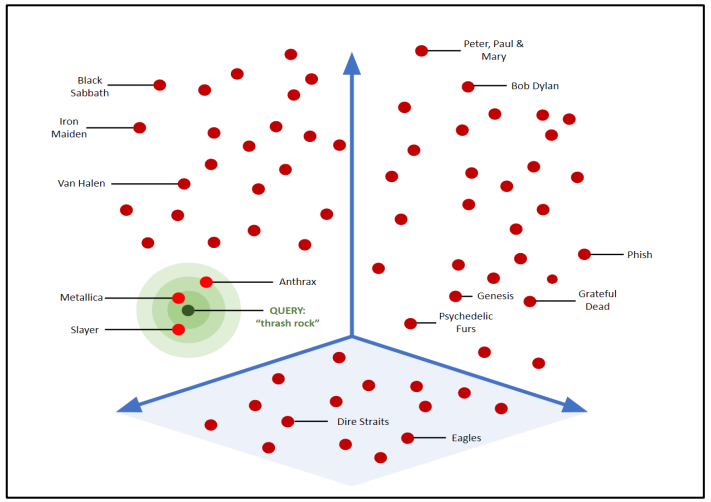


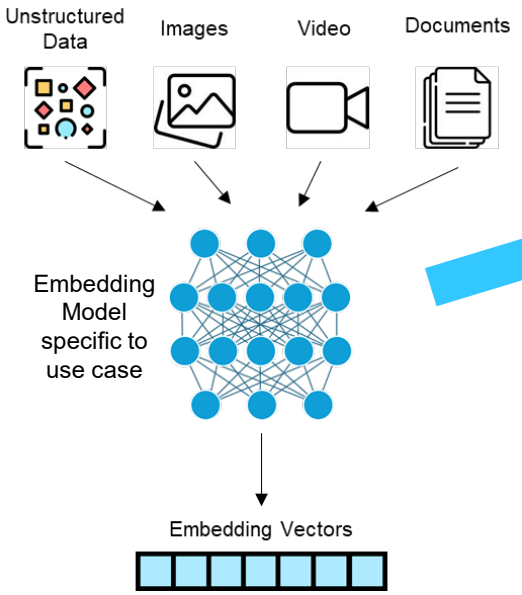
Vector Database

Vector Database – Contextual Similarity

- Vector Databases use Embedding Vectors to fuel Approximate Nearest Neighbor searches (ANN)
  - Embedding Vectors are generated by AI Models tuned to map data context to a vector space
  - In the vector space, data that is contextually similar will be “near” each other
  - Embedding Models are developed for specific use cases

Vector Space





LLM RAG

- Find contextually relevant documents to provide data to LLM queries. More & better data with queries results in more intelligent AI
- Used to enable *other* AI models

Recommendation Systems

- Recommendations for social media, online shopping, ads, search results
- Used at scale

Fraud Detection

- Details of transactions get embedded into a vector then compared to the database of past transactions of valid and fraudulent. High % of similarity to past fraudulent transactions = DECLINE
- Efficiency & Quality of search results as direct monetary effect

Vector Databases are much broader than just feeding LLM models

Vector Database – Indexing

Building a vector index is memory intensive with many small IOs

ANN Indexes

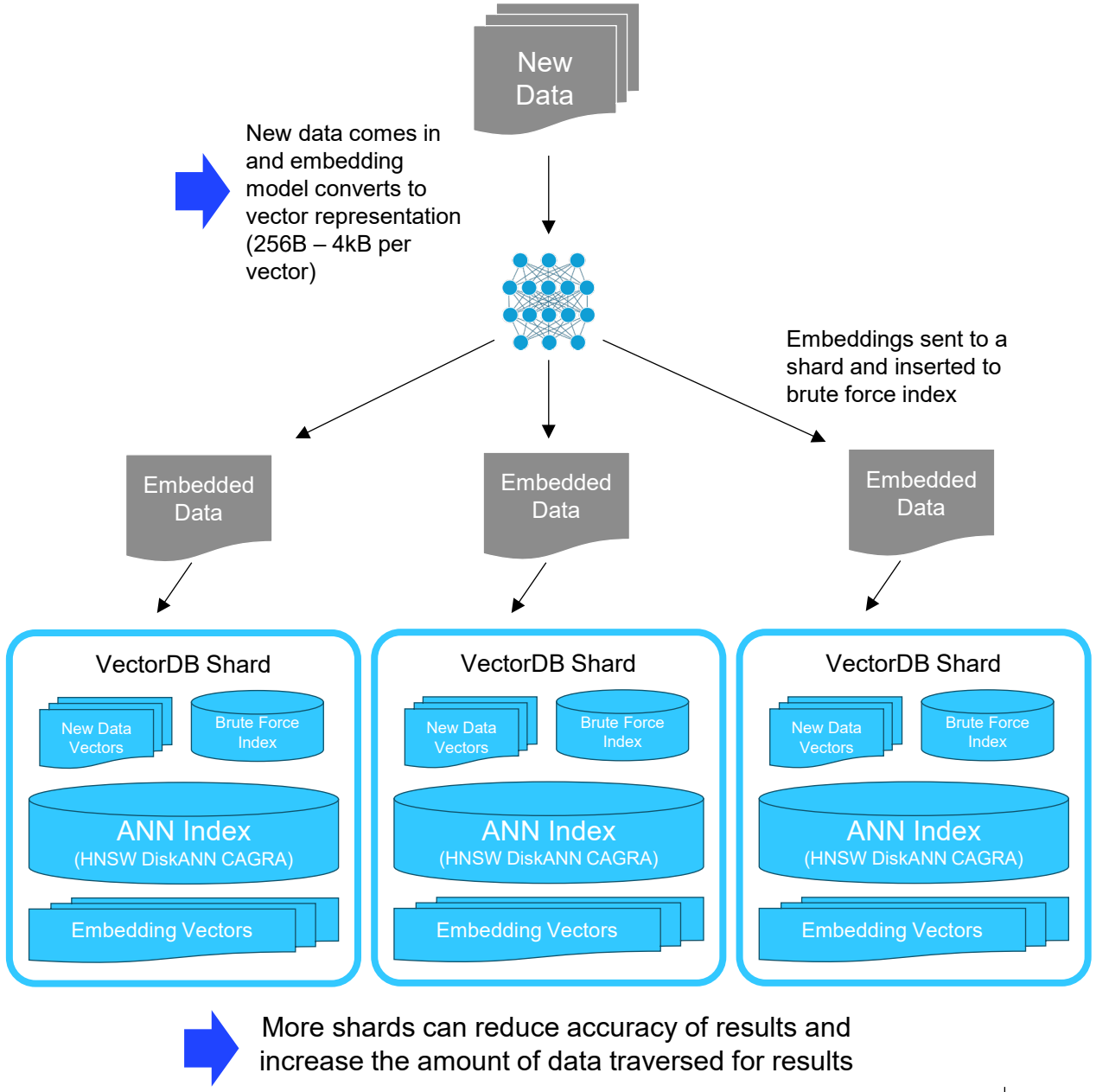
- Tuned for query performance and accuracy
- Require rebuild of entire shard when new data crosses a threshold (~2%) ->40-50 times/day

ANN Index building

- Uses quantized version of vectors to reduce size
- Random small access across the dataset to build the index
- 1B Vectors ~= 100GB – 1TB Index
- Shard sizes effectively dependent on memory per node

Indexing infrastructure choices

- CPUs (10x memory) -> slow build with fewer shards
- GPU (30x perf) -> fast build with more shards



ANN: Approximate Nearest Neighbor

Would you like to know more?

4:05-4:35pm: DiskANN - VectorDB Indexing Offload

- Alessandro Goncalves – Solidigm – Solutions Architect

Vector Database – Benchmarking Challenges

Data

- Inputs to vector database are embedding vectors
  - Have a dimensionality (eg: 128 to 2048 dims)
  - Have a data type (Int8, FP8, FP16, BFLOAT16, etc)
- Available datasets can be raw data or pre-calculated embedding vectors
  - Limited in size (100M to 5B vectors)
  - Limited in type (eg: only 768 dim @ FP16)
- Creating embedding vectors is computationally intensive (generally run on GPUs)

Tools

- One primary existing tool – vectordb-bench
  - https://github.com/zilliztech/VectorDBBench
- Focused on measuring performance ***and*** accuracy of results
  - No clear methodology to scale dataset size beyond the given size of the real datasets
  - Requires generating a ground truth nearest neighbor list for queries (computationally intensive)

Databases / Indexes

- Indexes and database software effect performance AND accuracy
- HNSW can use storage via MMAP
- DiskANN natively uses storage

MLPerf Storage – Vector Database

End to End Vector Database Benchmark

- Create database with random data
  - Enables scaling size of databases without being dependent on size of available datasets
- Creates ground truth for recall accuracy measurements
- Manage index creation & compaction
- Run concurrent query streams of batched queries
- Measure QoS Latencies and query throughputs
- Tooling for increased DB & Index support in development
  - HNSW, DiskANN, AISAQ, FLAT, IVFFLAT
  - Milvus, pgvector, Elasticsearch

2.2 Production Use Cases Driving Storage Demand

Use Case	Typical Scale	Dimension	I/O Pattern	Example
RAG for Enterprise LLMs	10M–1B+ chunks	768–1536	Read-heavy, random 4–64 KB	ChatGPT Enterprise, Claude with internal docs
E-commerce Product Search	100M–500M products	256–512	Read-heavy, bursty	Amazon product recs, Shopify vector search
Content Moderation	1B+ embeddings	512–768	Read-heavy, sustained	Trust & safety scanning uploads
Fraud Detection	10M–100M signals	128–512	Write-heavy ingest, read-heavy query	Fintech real-time fraud scoring
Code Search / Copilot	50M–500M chunks	768–1536	Read-heavy, multi-turn	GitHub Copilot, Cursor, internal code search
Multi-Modal Search	10M–100M images	512–2048	Read-heavy, large vectors	CLIP-based image search, audio fingerprinting
Autonomous Driving Maps	100M+ map tiles	128–256	Read-heavy, geospatial	Tesla, Waymo map tile similarity

KV Cache Management

Would you like to know more?

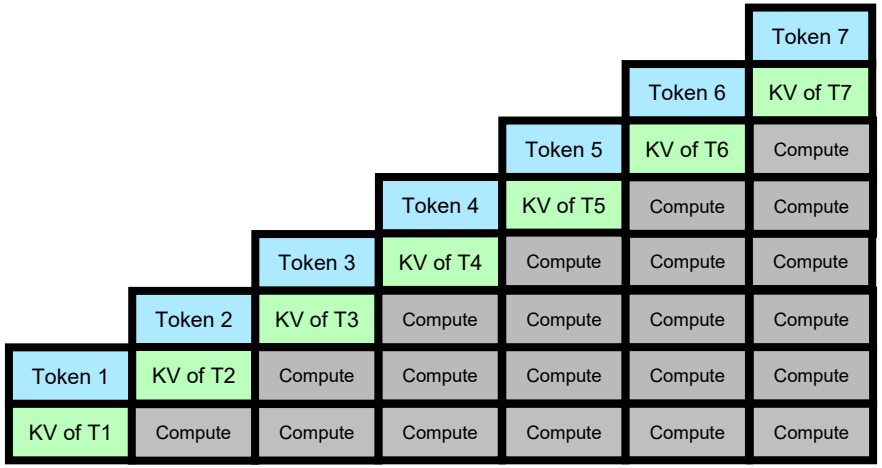
2:10-2:40pm: Scaling Inference with KV Cache Storage Offload and RDMA Accelerated Architecture

- Ugur Kaynar – Dell – Distinguished Engineer, Storage Technologist – Storage CTO

KV Caches – LLM's understanding of Words in Context

- Processing a Query
  - Content is converted to a series of tokens.
  - Tokens are converted to vectors with an embedding model
  - LLM generates K & V vectors for each token based on K & V vectors of prior tokens ("attention")
  - LLM "Caches" the K & V vectors from prior tokens for processing the next token. The "KV Cache" is internal to the model.
  - As queries get longer:
    - KV Cache size grows linearly – O(n)
    - Computation *per token* increases
    - KV generation compute requirements for a query scales ***quadratically*** with length – $O(n^2)$
  - Latest models support context lengths = 1 million + tokens
    - Calculating the KVs for ~1M tokens on 8x B200s for Llama4 Maverick in FP8
      - 140 seconds
        - Model can not generate new tokens until this phase completes
        - Time to first Token (TTFT) is more than 2 Minutes
      - **15.6 GB of KV Cache** for this single query
      - https://www.micron.com/about/blog/company/insights/1-million-token-context-the-good-the-bad-and-the-ugly

★ KV Cache uses Memory Capacity to reduce Compute Requirements during token generation (decode)

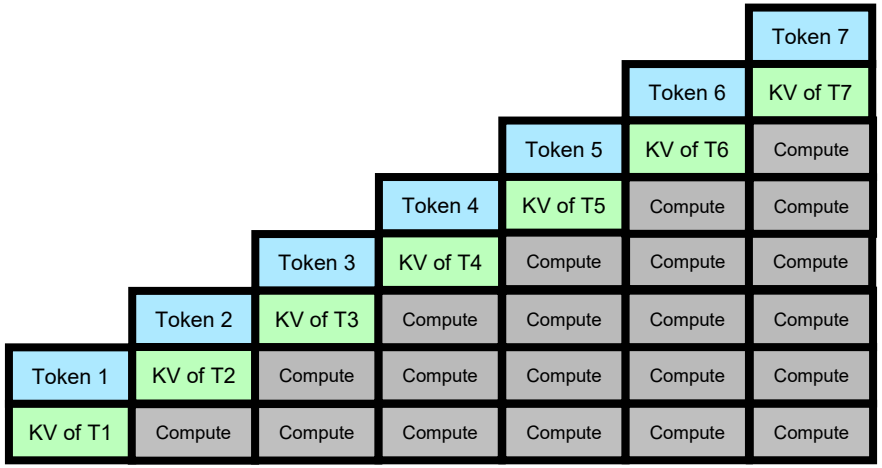


KV Caches – Offloading vs Reuse

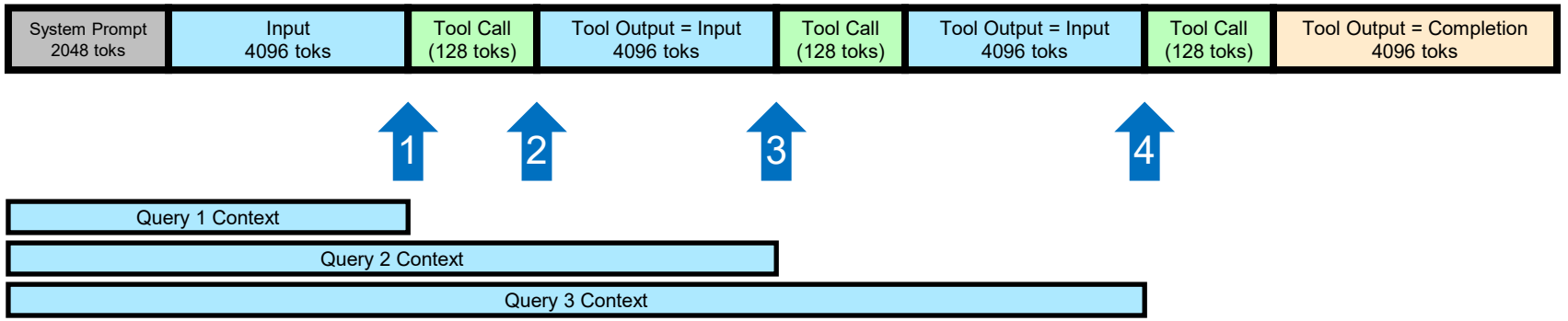
Two techniques used with KV Caches:

- KV Cache ***Offload*** accesses KV cache ***during decode*** for each token generated.
  - Requires moderately high bandwidth per GPU (>200GB/s)
  - CPU DRAM or CXL Memory or *very* fast storage

- KV Cache ***Reuse*** accesses KV cache ***once*** at the start of token generation.
  - Requires storage that is faster than the time to recompute the KV Caches or *fast enough to meet query SLAs*



Agentic Systems driving KV Cache Management

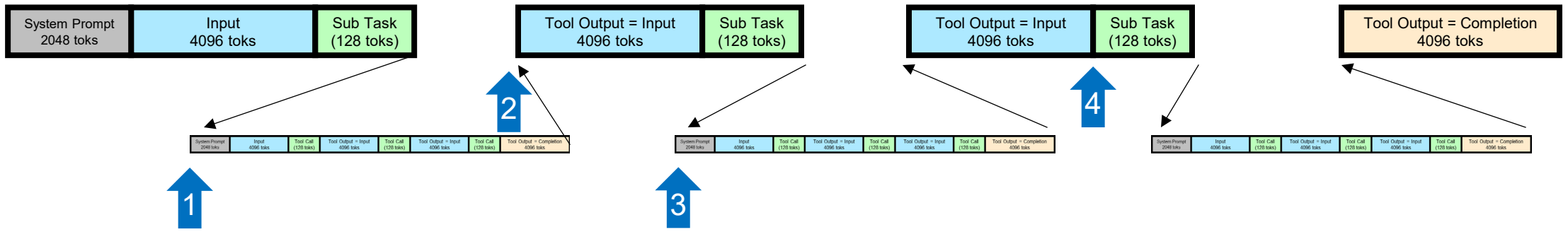


1. First query, send System Prompt, input query, additional context (code, documents, etc)
2. Output is a tool call that gets executed
   Read document, run command, execute REST API call, grep, git, find, etc
3. The output from a tool call gets added to the context (input) for the next call to the LLM
4. The original prompt, input query, tool call command, and output from tool call are the input for the next query.

Each successive turn in an agentic LLM conversation includes the entire history.

Any portion that was already run through the model is potentially served via caching

Agentic Systems driving KV Cache Management



1. A tool call can be to spawn a subtask
2. The Orchestrator session is paused while the subtask executes (minutes to hours)
3. A single orchestrator can spawn multiple subtasks as simpler tasks are handled by smaller models (less compute, less KV KB)
4. Orchestrator sessions generally use higher cost models (higher intelligence) but can have longer time between KV reuse.

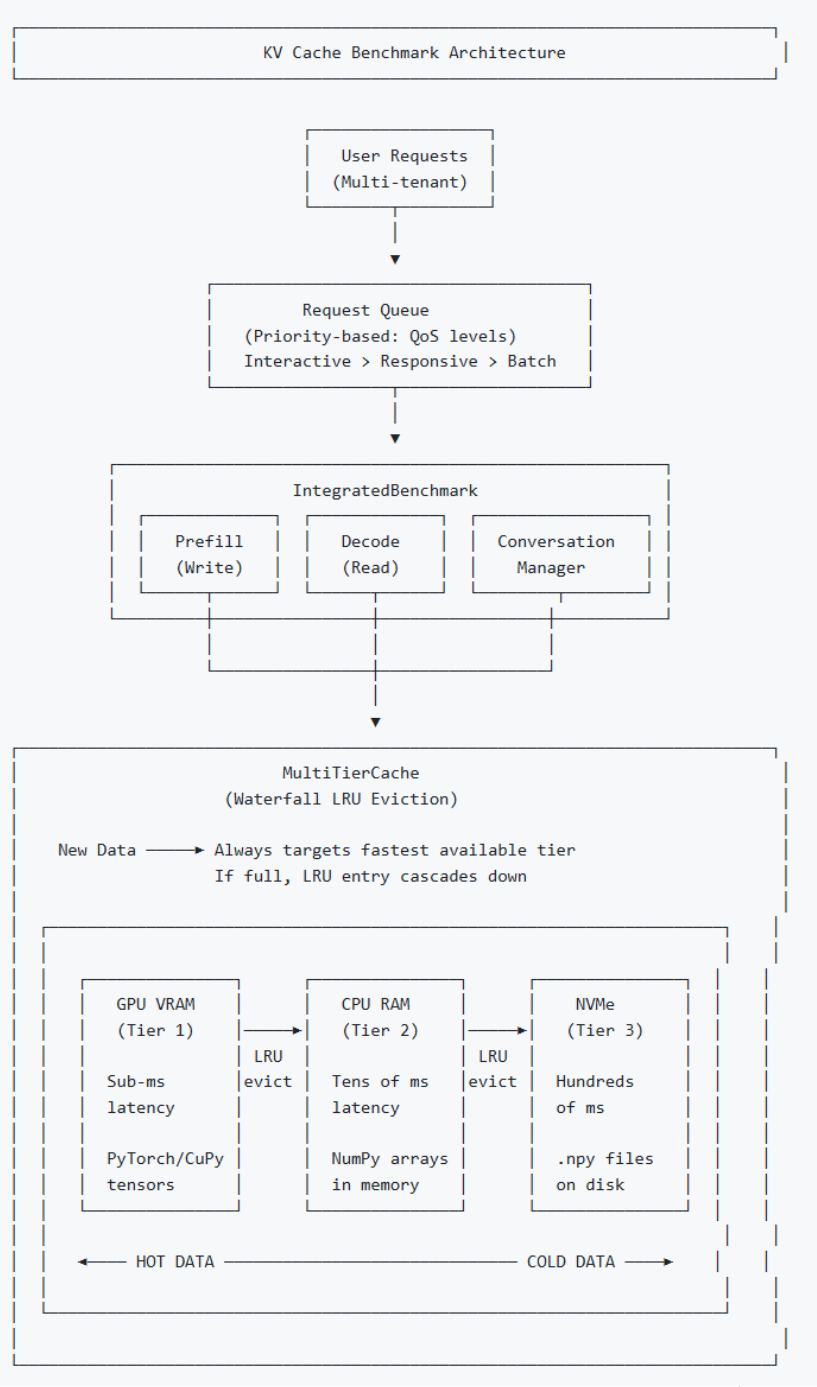
Not all Cache Hits have equal value

Simple LRU caching would incorrectly evict the orchestrator KVs due to longer temporal access pattern even though orchestrator KVs being cached provides greater TCO savings

MLPerf Storage – KV Cache

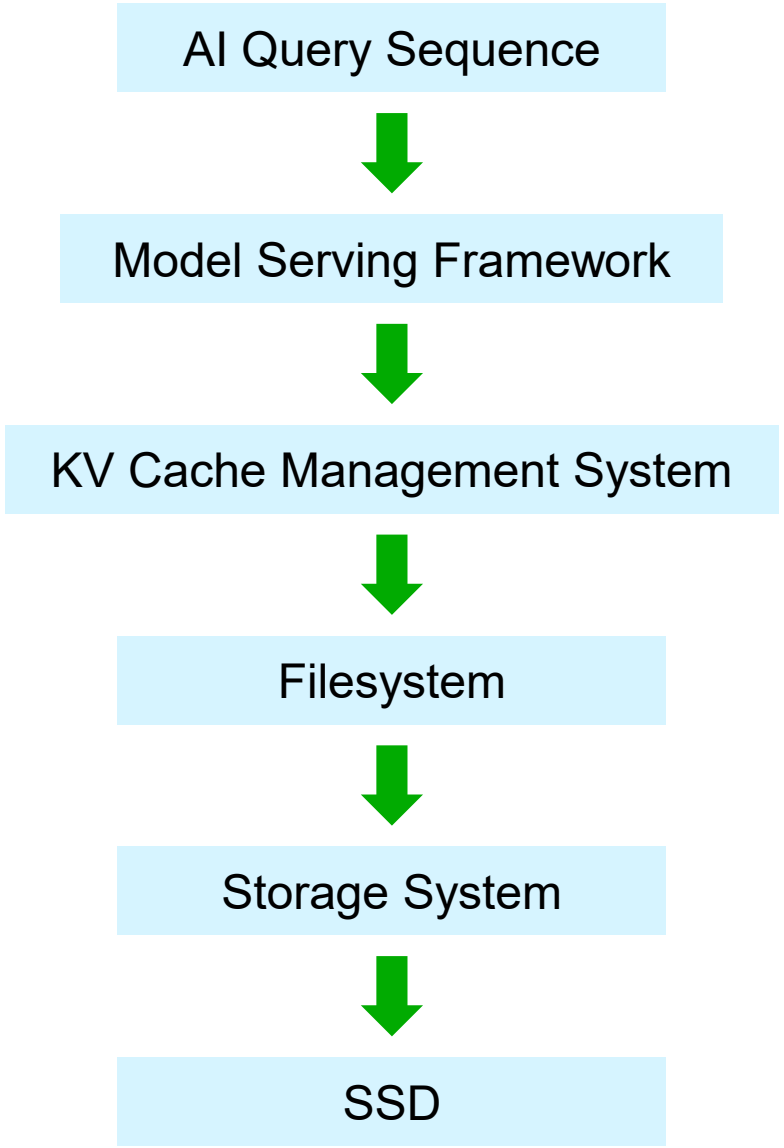
Self-contained KV Cache Management Simulation Benchmark

- Manages KV chunks across tiers
  - GPU Memory (not required for operation)
  - CPU Memory
  - Storage
- Supports representing a range of models with varying KV KB per token
  - 20KB – 500KB
- Scale concurrency to QoS limit to measure supported query throughput for various query patterns
  - Chatbot, Coding, Document interaction



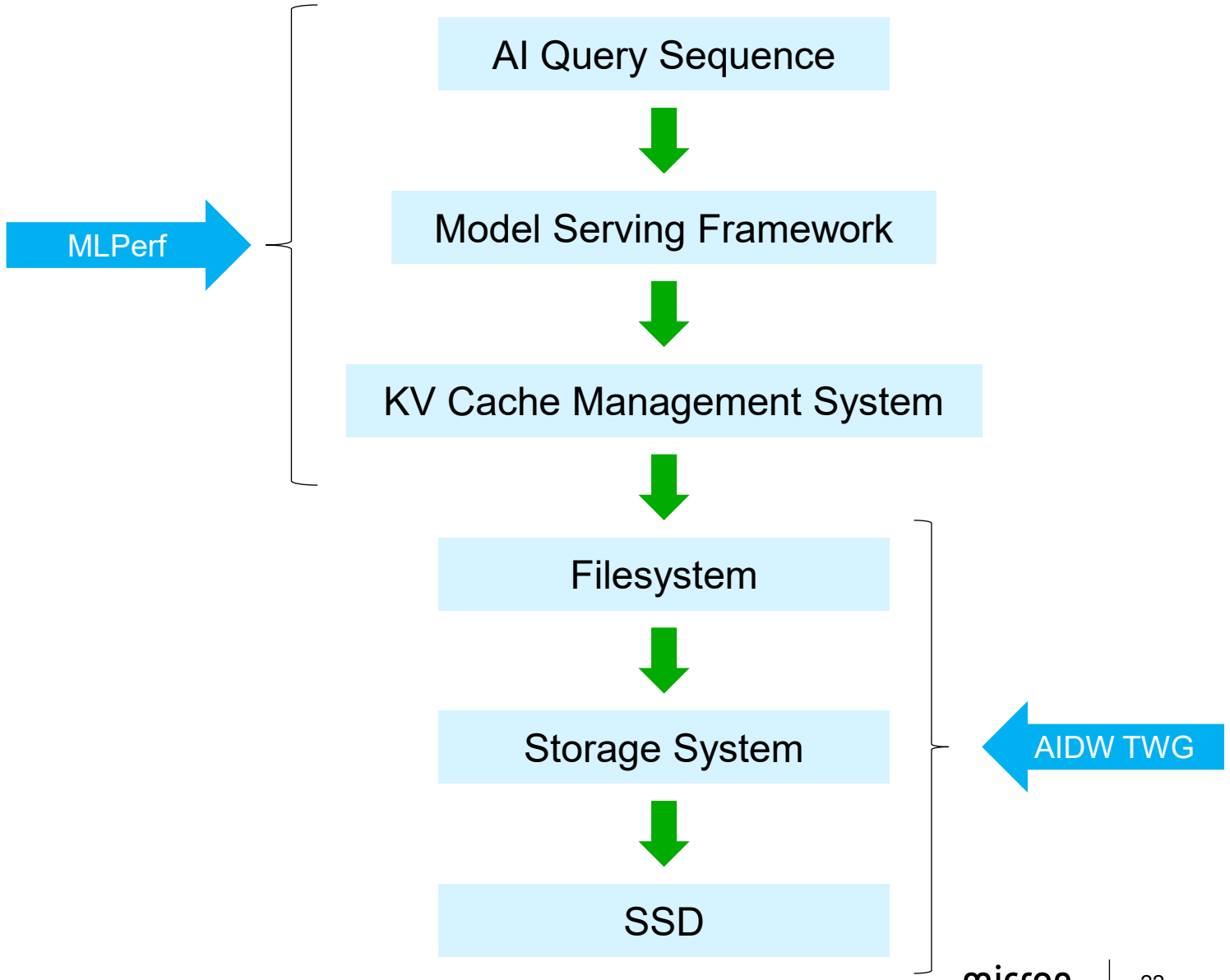
SNIA AI Data Workloads TWG – KV Cache Analysis

- Mapping an LLM workload (query sequence) to a physical device's IO pattern
  - Dependent on multiple layers
  - Model serving framework
    - TensorRT
    - SGLang
    - vLLM
  - KV Cache Management System
    - Dynamo KVBM
    - LMCache
    - Mooncake
  - Filesystem
    - Closed Source Parallel FS
    - Lustre
    - XFS/EXT4
  - Storage System Software / Hardware
    - Erasure Coding
    - Data distribution
  - Physical Devices
    - MDTS
    - IU



SNIA AI Data Workloads TWG – KV Cache Analysis

- Where is the best place for our work to sit?
- MLPerf Storage aims to provide an end-to-end benchmark that generates a query sequence and directly executes a workload to underlying mounted filesystems.
- AIDW-TWG aims to define a synthetic workload that can be executed to a filesystem, object store, or block device.



micron Intelligence Accelerated™