THE AI DATA COMPANY

Presenters:
IZHAR SHARON
NASIR WASIM

Our Journey

1998

The Beginning

1998-2008

Leadership in High Performance Computing

2008-2018

Expansion in Enterprise Markets & Technological Innovation

2018-2023

Accelerated Growth & Leadership in AI Analytics, LLM Training & Gen AI

2024-Present

Go-to Leading Force in AI Data Intelligence

Accelerating AI

How Modern Object Storage Transforms RAG Performance

Samsung Conference Session

Optimizing Infrastructure for Enterprise AI Workloads

Introduction to RAG (Retrieval-Augmented Generation)

- The problem with LLMs: Hallucination & Static Knowledge
- What RAG adds: Live retrieval for factual grounding
- Why this matters for AI-powered enterprise solutions
- The role of efficient retrieval in AI accuracy

RAG Technical Components

- **Tokenization:** Converting raw text into processable units for NLP models
- **Embeddings:** Transforming text tokens into high-dimensional numerical vectors that capture semantic meaning
- **Vector Databases:** Specialized systems (like Milvus/FAISS) for storing embeddings and performing similarity searches at scale
- **Chunk Storage:** Object storage systems that hold the actual document content for retrieval after vector matching
- **Similarity Search:** Mathematical operations (cosine similarity, dot product) to find semantically related content

What You're Experiencing

- RAG systems work perfectly in development
- Production deployments struggle under real user load
- Response times degrade with concurrent users
- Infrastructure costs spiral unexpectedly

Everyone focuses on LLMs and vector databases.

The bottleneck is actually in object storage.

Retrieval Augmented Generation (RAG)

User Query
"How do I configure X?"
→
Vector Search
Find relevant chunks
→
Retrieve Context
Get full text content
→
LLM Response
Generate answer with context

- Solves LLM knowledge limitations without expensive fine-tuning
- Enables AI to access current, proprietary information
- Allows dynamic knowledge updates without model retraining
- Critical for enterprise AI applications

Enterprise RAG Processing Pipeline

Enterprise CMS
Internal document systems
SharePoint, Confluence, etc.

→

Text Extraction
Content parsing
PDFplumber, OCR

→

Smart Chunking
500-token segments
LangChain splitters

→

Dual Storage
Performance comparison
AWS S3 + DDN INFINIA

Vector Embeddings
FAISS indexing
SentenceTransformers

→

Similarity Search
Query matching
Vector similarity

→

Context Retrieval
Object storage fetch
TTFB measurement

→

LLM Response
Nvidia AI generation
OpenAI, Nvidia, OSS

Tech Stack

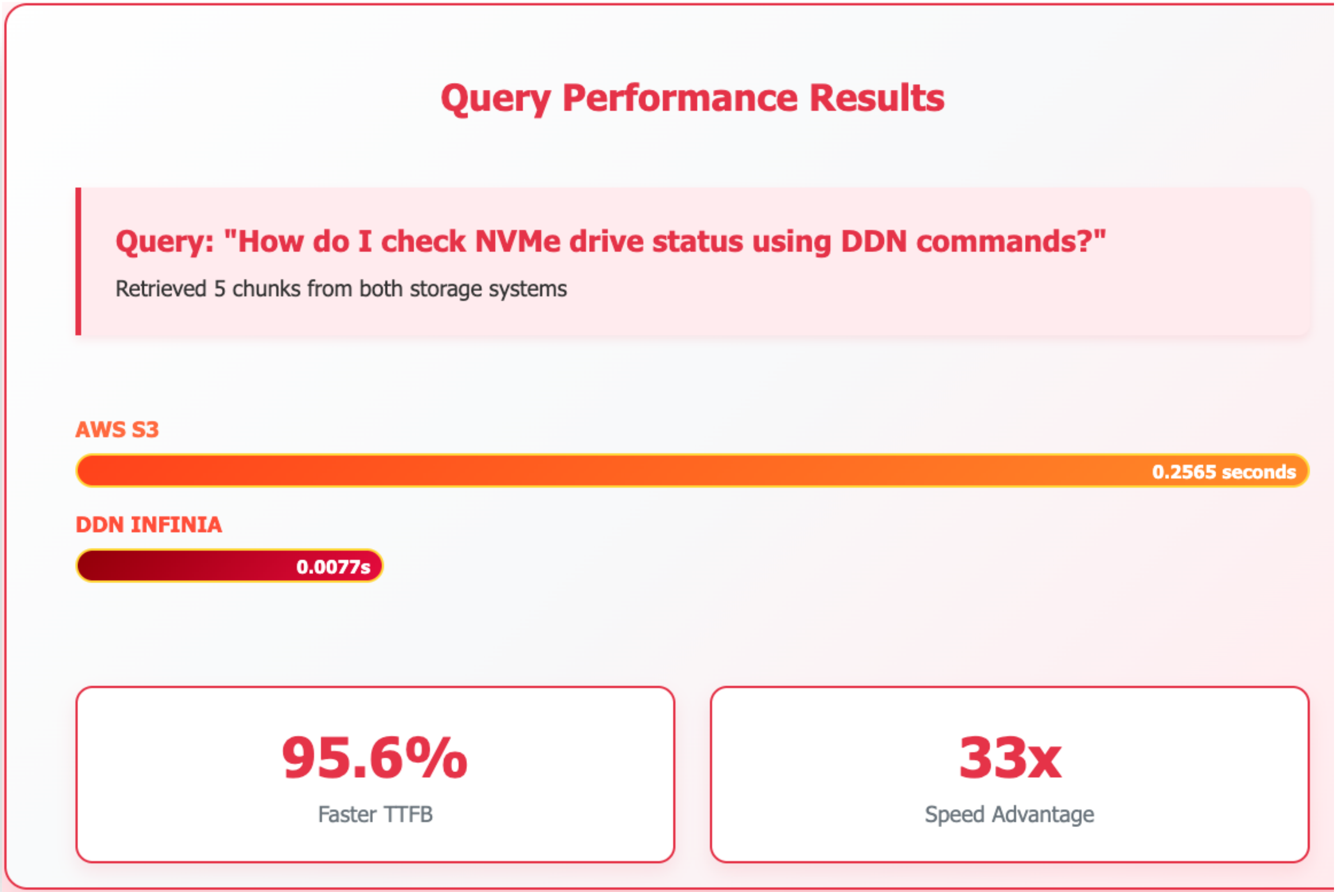
Storage: DDN INFINIA vs AWS S3 • **Vector DB:** FAISS • **LLMs:** Nvidia AI, OpenAI, Open Source • **Framework:** LangChain, Gradio

Live Demonstration

Real-World RAG Performance Test

Identical chunks stored in AWS S3 and DDN INFINIA simultaneously

- Complete document processing pipeline
- Actual network calls, authentication, serialization
- Same data, same conditions, different storage backends
- Production-realistic workload patterns



Query Performance Results
Query: "How do I check NVMe drive status using DDN commands?"
Retrieved 5 chunks from both storage systems
AWS S3
0.2565 seconds
DDN INFINIA
0.0077s
95.6%
Faster TTFB
33x
Speed Advantage

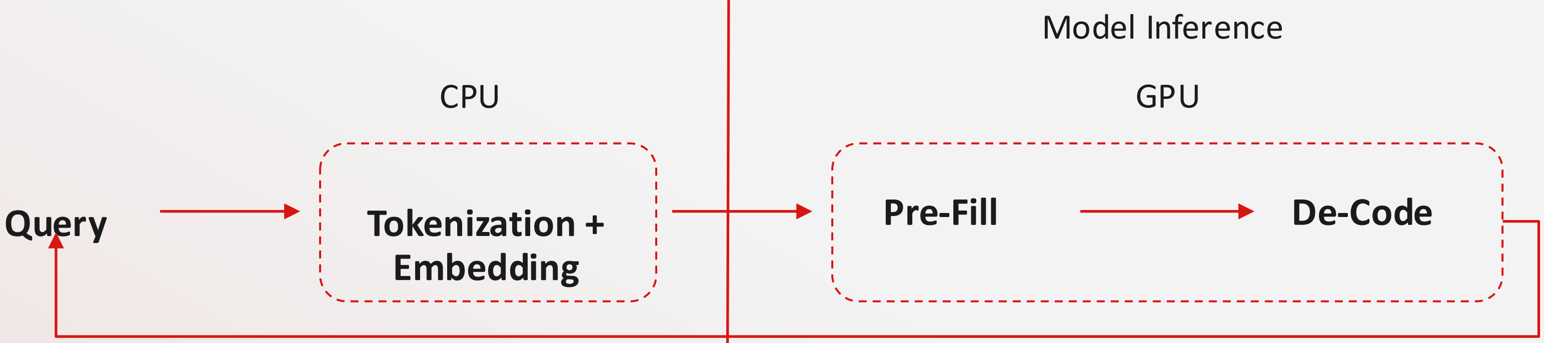
ddn

The World's Leading
Data Intelligence Platform

KV CACHE – INTRO

[ANY DATA] [ANY APPLICATION] [ANYWHERE]

Day in the life of a Prompt



The user sends a prompt

Additional data is retrieved or injected to influence model response.

RAG

Process the entire input context to generate K/V Cache

Vectorization

Attention calculation

This is **quadratic** in cost

Most expensive when prompt is long

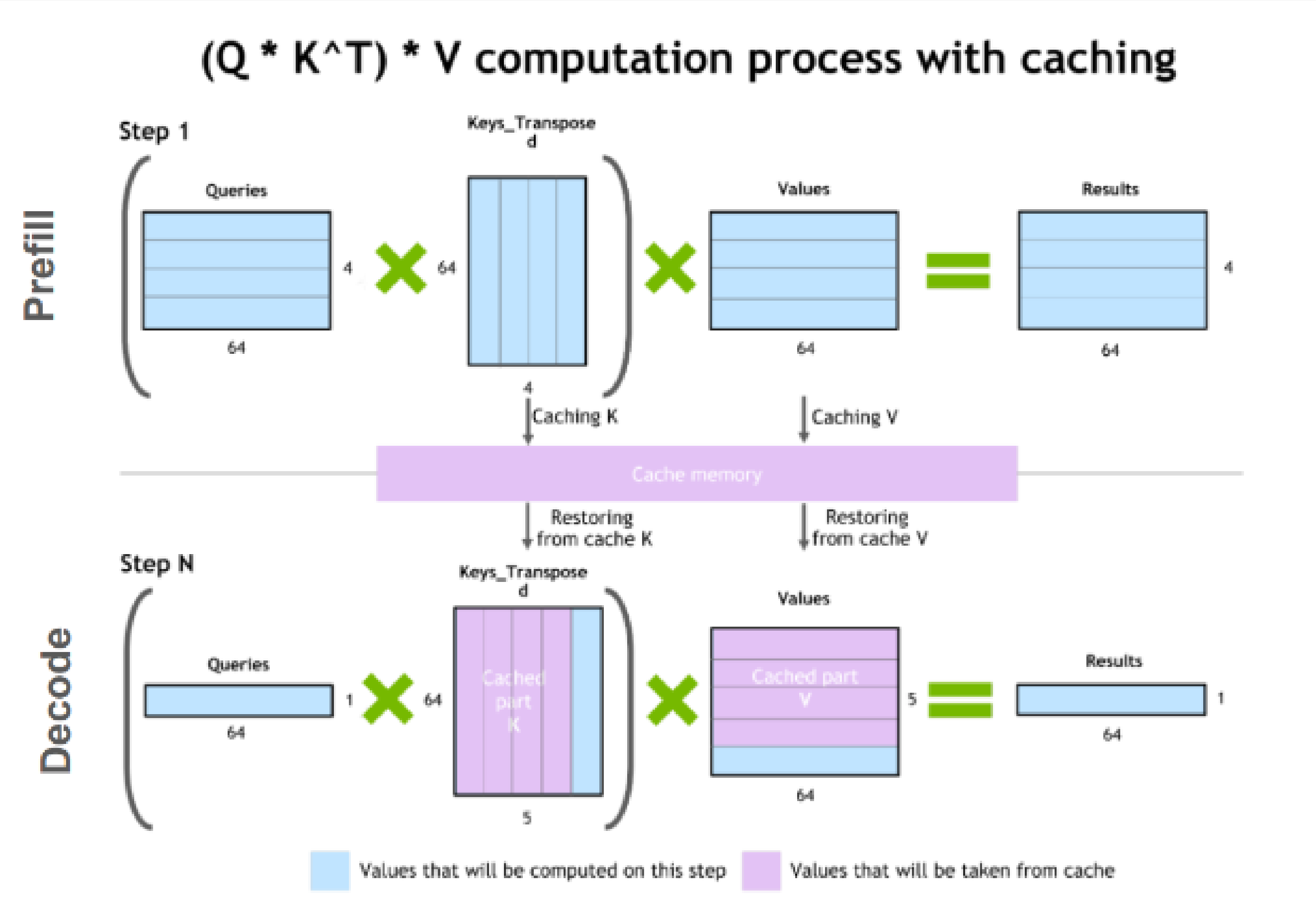
Model Generates Tokens

token-by-token generation

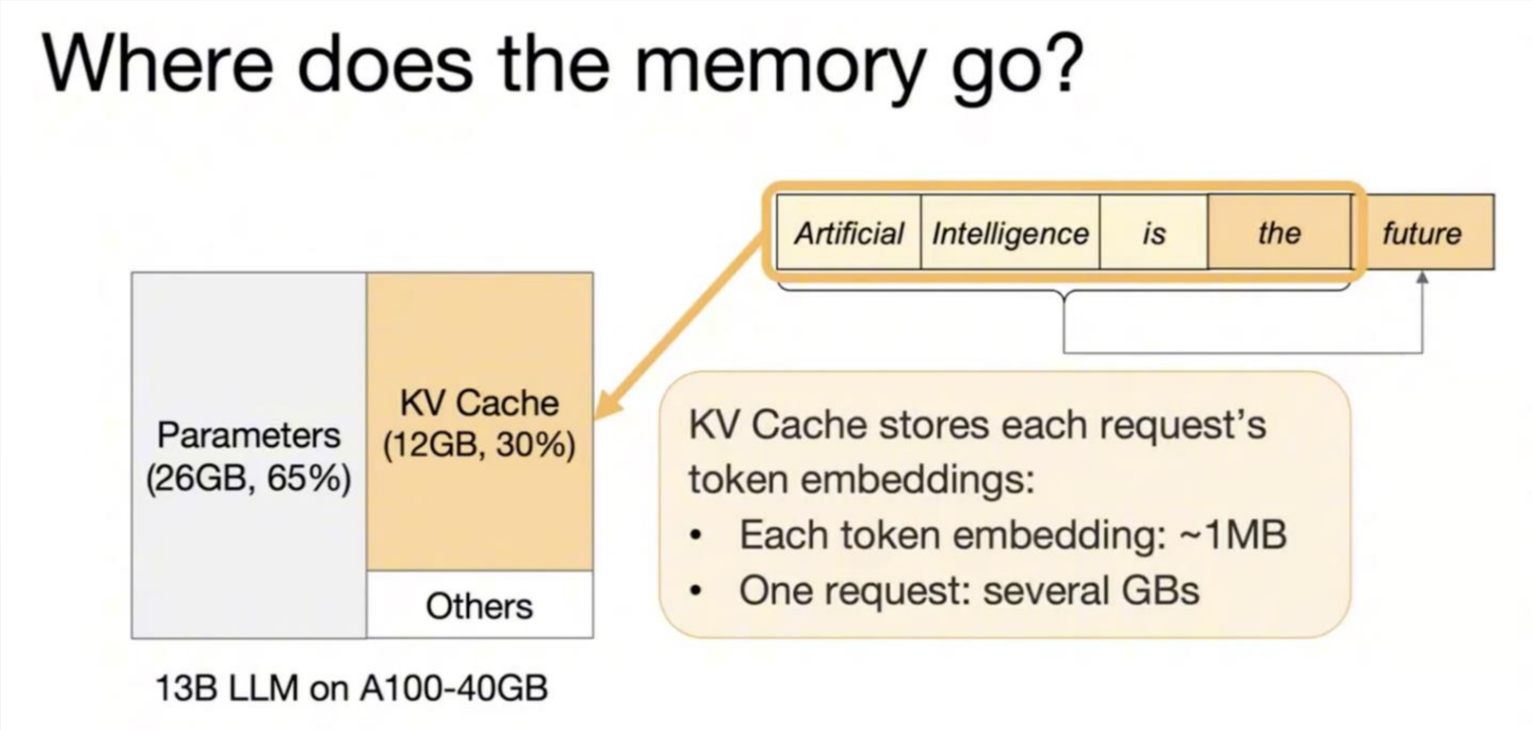
Linear in cost

Attention updated

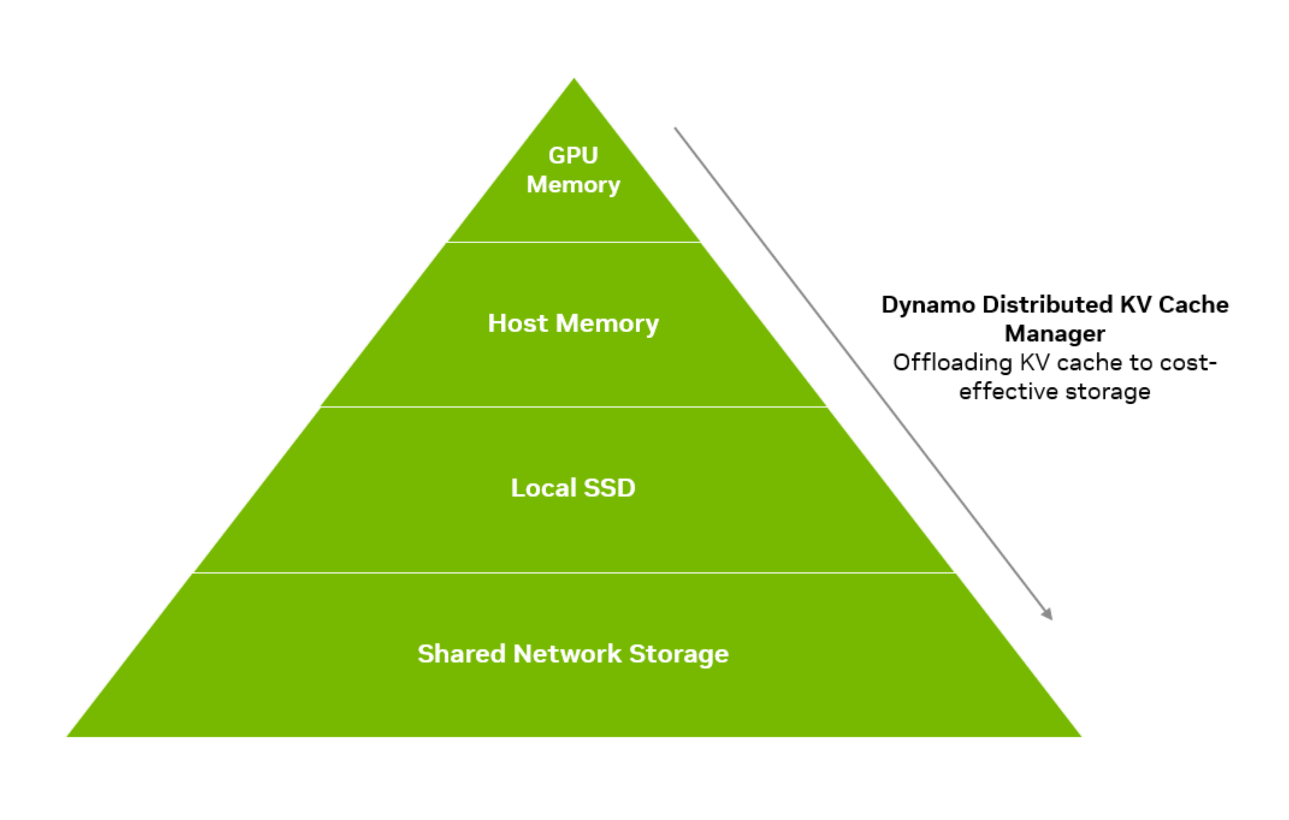
Key Value saved in memory to De-code only new tokens



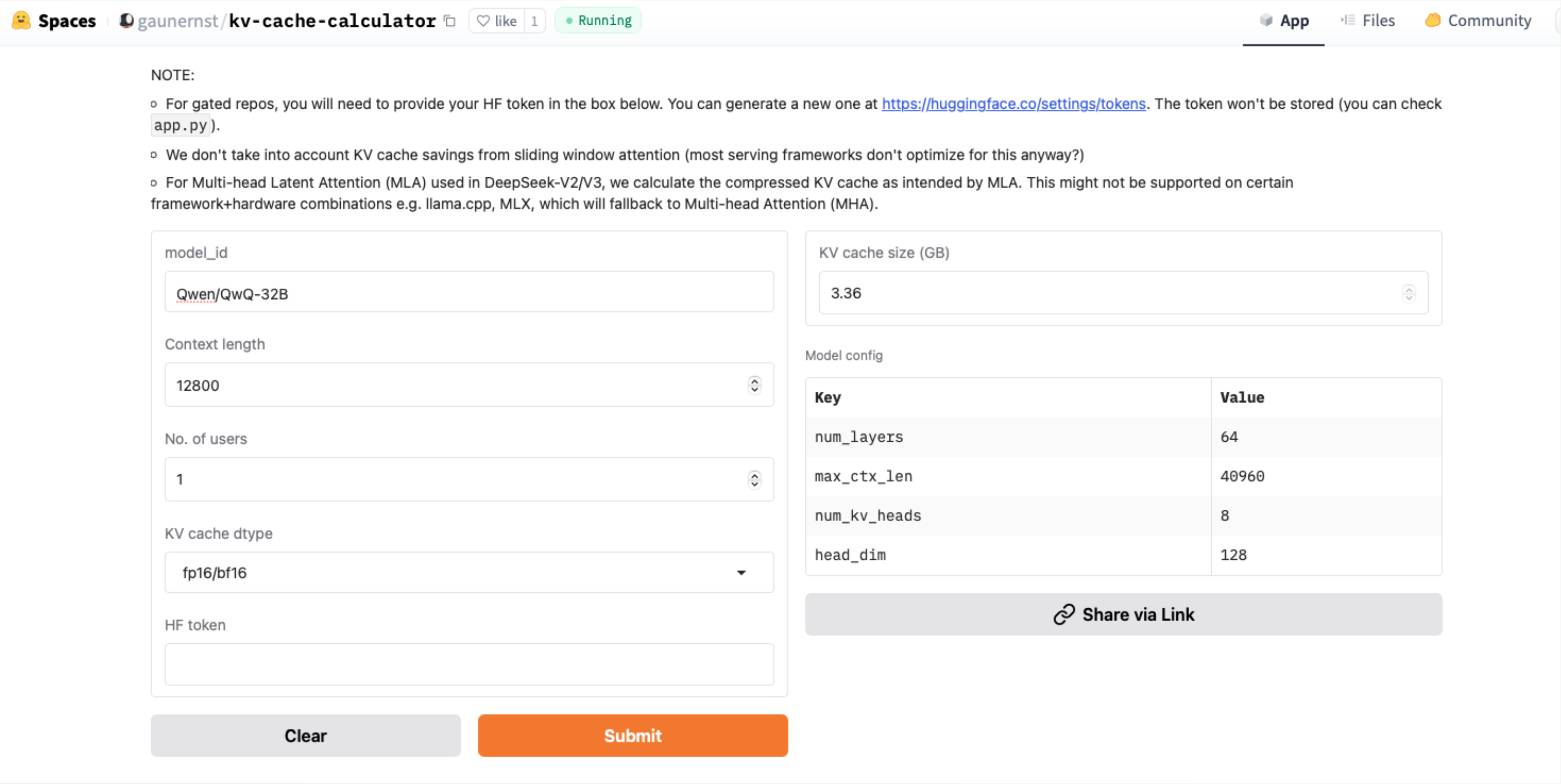
KV Cache is a bottleneck for Inference



NVIDIA Dynamo Distributed KV Cache Manager



KV Cache Calculator
https://huggingface.co/spaces/gaunernst/kv-cache-calculator



What gets cached?

The cache stores:

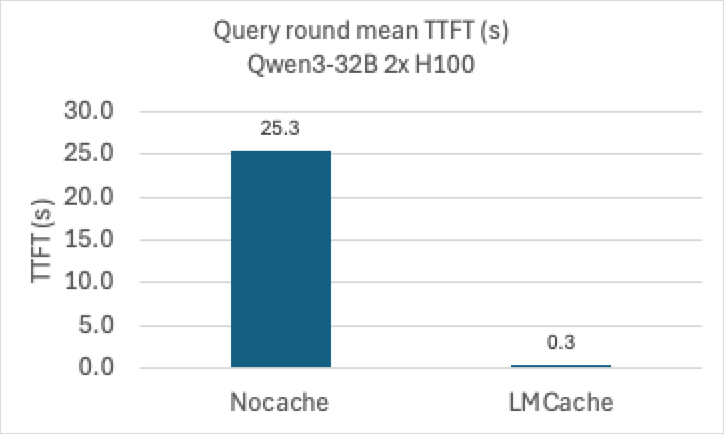
- **Key**: Token ID sequence (the matching identifier)
- **Value**: The computed key-value tensors from the attention layers for those tokens

So the workflow is:

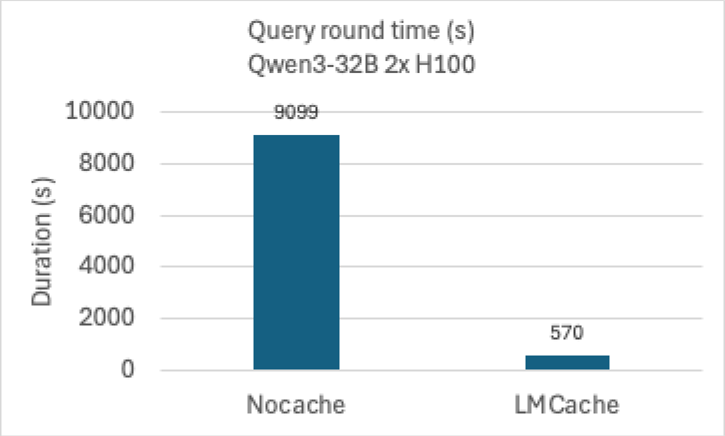
1. Input text → Tokenization → Token ID sequence
2. Match token IDs against cache keys
3. On hit: Retrieve the pre-computed KV tensors for those tokens
4. On miss: Run inference and cache the resulting KV tensors with the token sequence as the key

The cached KV tensors represent the internal attention states that would have been computed if those tokens were processed fresh - this is what enables skipping the expensive forward pass computation for the matched prefix.

DDN KV Cache Improvement - Qwen3-32B - 2x H100 GPUs



75.6x improvement



16x improvement

Scenario:

- Warm-up round: Send 100 documents of 130K tokens to the engine
- Query round: Send 4 questions about each document (400 prompts in total)
- Measure: mean TTFT across queries; total duration for the query round

ddn