dino_loader.py — NVL72 Cluster

```python
# HPC-Grade DINOv3 Data Pipeline
# Target: B200 / GB200 NVL72 Clusters

class DinoLoader:
```

dino_loader

```python
    status = "Production-Ready"
```

```python
def __init__(self):
    """
    Designed for self-supervised vision model training at petascale:
    hundreds of GPUs, hundreds of millions of images.
    GPU training is *never* bottlenecked by data ingestion.
    """
```

main

0 errors, 0 warnings

UTF-8

Python 3.12

stderr.log — Performance Monitor

⚠ CRITICAL_FAILURE

[10:00:01] **FATAL** CPU JPEG Decoding saturated at 1,000 img/s. Required: 50,000+.

[10:00:02] **ERROR** PCIe 5.0 Bandwidth bottleneck. Transfer stalled at 64 GB/s cap.

[10:00:03] **WARN** Lustre Metadata Latency High. Sequential reads: ~1 shard/sec.

[10:00:04] **FATAL** Redundant I/O Detected. 72 ranks reading same shard. Network Load: 7200%.

GPU UTILISATION vs DATA INGESTION

Target Throughput 50,000 img/s

TARGET

Actual Throughput 1,000 img/s

CRITICAL

GPU STARVATION: 98%

Compute units idle waiting for data.

System Halted PID: 4192 Mem: 128GB

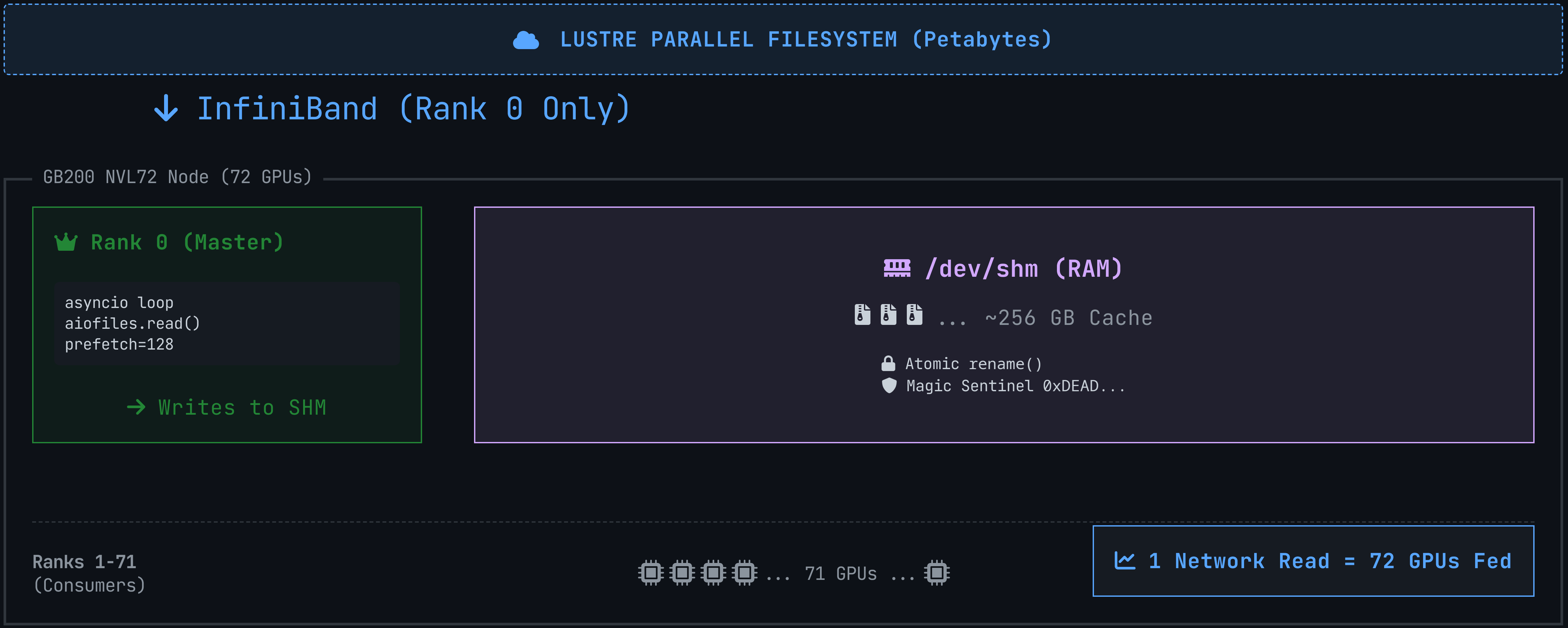
Concurrent Pipeline Architecture

STAGE 1	STAGE 2	STAGE 3	STAGE 4	STAGE 5
Shard I/O	**JPEG Extraction**	**Augmentation**	**H2D Transfer**	**FP8 Quantisation**
asyncio + aiofiles /dev/shm	ThreadPoolExecutor memoryview	NVIDIA DALI nvjpeg ASIC	AsyncPrefetch CUDA Stream	Transformer Engine BF16 → FP8
10-50 GB/s bandwidth One read per shard	Zero-copy parsing 2 shards in-flight	>3× CPU throughput Fused GPU kernels	Non-blocking Overlapped compute	Zero conversion cost Direct TE input

Fully Asynchronous Execution

Every stage runs concurrently. No stage waits for the next. The only back-pressure is intentional to prevent memory overflow.

Multi-Node Data Flow



gpu_pipeline.cu — NVIDIA DALI

GPU-Native Augmentation @ Hardware Peak

```python
# DINOv2 Augmentation Protocol (Fused Kernel)
def build_pipeline():
    # 1. Hardware Decode
    imgs = fn.decoders.image(device="mixed")

    # 2. Geometric Transforms
    imgs = fn.random_resized_crop(imgs, size=224)
    imgs = fn.flip(imgs, horizontal=1)

    # 3. Color Ops (BF16 Precision)
    imgs = fn.color_twist(imgs, hue=0.1, sat=0.2)
    imgs = fn.gaussian_blur(imgs, window_size=23)
    imgs = fn.solarize(imgs, threshold=128)

    # 4. Output Formatting
    return fn.transpose(imgs, perm=[2, 0, 1])
```

HARDWARE ACCELERATION

nvjpeg ASIC

Decodes JPEGs directly on-chip. >3× throughput vs CPU. Keeps PCIe bus free.

PRECISION

BF16 Tensor Cores

Native Blackwell support. Safe for 0-255 pixel math. No overflow risk.

CROPS STRATEGY

2 Global + 8 Local

10 crops per image generated in parallel.

SINGLE FUSED KERNEL PASS

Dynamic Curriculum Learning

```
specs = [ DatasetSpec("laion2b", weight=0.5),
DatasetSpec("datacomp1b", weight=0.3),
DatasetSpec("imagenet22k",weight=0.2), ] # Training Loop for
epoch in range(100): # Dynamic curriculum shift at epoch 10 if
epoch == 10: # Thread-safe, effective next batch
loader.set_weights([0.4, 0.4, 0.2]) # Re-shuffles shards with
new seed loader.set_epoch(epoch)
```

> **Performance Note**
>
> Uses random.choices(k=batch_size).
> Complexity is **O(k)** — single call selects all indices.

WEIGHT DISTRIBUTION (LIVE)

laion2b	50% → 40%
datacomp1b	30% → 40%
imagenet22k	20%

RESHUFFLING LOGIC

Seed Formula:

base + rank + epoch * 1_000_003

- ✓ Reproducibility
- ✓ Per-rank diversity
- ✓ Per-epoch diversity

checkpoint.json | nccl.conf

JSON State Serialization

dl_state_000001000.json

```json
{
  "step": 1000,
  "epoch": 5,
  "weights": [0.5, 0.3, 0.2],
  "datasets": ["laion", "datacomp"]
}
```

- **Stable & Portable**
  JSON > Pickle. No class metadata dependency. Safe across heterogeneous environments.
- **Atomic Writes**
  Writes to .tmp, then POSIX rename(). Rank 0 only.
- **Resume Logic**
  Restores epoch & weights. DALI pipeline state reset (sub-epoch resume via DALI 1.30+).

NVL72 Fabric Tuning

NCCL_IB_DISABLE	1	Disable InfiniBand. Force NVLink-C2C usage.
NCCL_NVLS_ENABLE	1	Enable NVLink Switch reductions (4x faster).
NCCL_PROTO	LL128	Low-latency 128-byte protocol for NVLink.

Health Check

verify_interconnect() runs a canary all-reduce at startup to detect degraded links before training begins.

Data Journey: Lustre → VRAM

01	**Lustre → /dev/shm**	Async I/O download to shared tmpfs RAM. One read per node.	RAM
02	**/dev/shm → memoryview**	mmap() creates a process-level view. **Zero-copy.**	RAM
03	**memoryview → deque**	bytes() slice extraction. **The only CPU-side copy.**	RAM
04	**deque → DALI CPU**	MixingSource callback. DALI takes ownership of buffers.	RAM
05	**DALI CPU → GPU** [H2D]	**nvjpeg ASIC** decodes directly into VRAM. Fused kernels follow.	VRAM
06	**GPU Queue → Training**	Exposed as torch.Tensor via shared pointers. **Zero-copy.**	VRAM

Architectural Efficiency

Once decoded in VRAM (Step 5), data **never returns to RAM.** The DALI GPU queue acts as a VRAM-to-VRAM ring buffer, pushing augmented batches directly to the training pass without PCIe round-trips.

Developer Ecosystem

Dataset Hub

Auto-Discovery

```
$DINO_DATASETS_ROOT/
  public/
    rgb/
      imagenet/
        shard-000000.tar
        shard-000000.idx
```

hub.py stubs generated for IDE autocomplete

Extensibility

CLI & Config

```
$ python -m dino_loader.datasets add private rgb my_new_dataset train
```

```
$ python -m dino_loader.datasets stubs
```

Custom Augmentation:
Subclass DINOAugConfig. Pipeline rebuilds automatically.

LIVE MONITOR (RICH UI)

JOB ID: 49210

Aggregate Throughput	Shard Cache Usage	Pipeline Stall Time
210,450 img/s	112 GB / 128 GB	0.0 ms

Reads from shared memory (lock-free). Zero overhead on training loop.

🚀 Quick Start & Installation

Installation

```
$ pip install nvidia-dali-cuda120 transformer-engine
$ git clone https://github.com/org/dino_loader && cd dino_loader
$ pip install -e ".[dev]"
```

Copy

SLURM Submission (GB200 NVL72)

```
# 4 Racks x 72 GPUs = 288 GPUs Total
$ sbatch --nodes=4 --ntasks-per-node=72 --gres=gpu:72 \
 --cpus-per-task=4 --mem=2048G --wrap="python train.py"
```

Copy

Key Configuration (LoaderConfig)

Parameter	Default	Tuning Guidance
node_shm_gb	128.0	Set to ~50% of node RAM for shard cache.
shard_prefetch_window	64	Increase for high-latency Lustre filesystems.
hw_decoder_load	0.90	0.90 saturates nvjpeg ASIC on Blackwell.
use_fp8_output	True	Disable only if Transformer Engine is unavailable.