Benchmarking AI Factories on MeluXina

A Modular Framework for Reproducible AI Workload Evaluation

Team 11

EUMaster4HPC Student Challenge 2025-2026

January 2026

The EUMaster4HPC Challenge

Challenge: *Benchmarking AI Factories*

How do AI workloads perform?

How to ensure reproducible benchmarks?

How to scale across SLURM-managed nodes?

Target: vLLM inference, MinIO S3, Vector DBs on MeluXina GPU partition

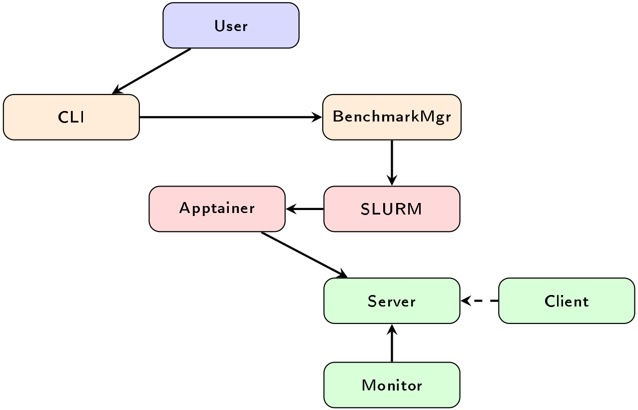
Our Solution: A Modular Benchmarking Framework

Three Pillars:

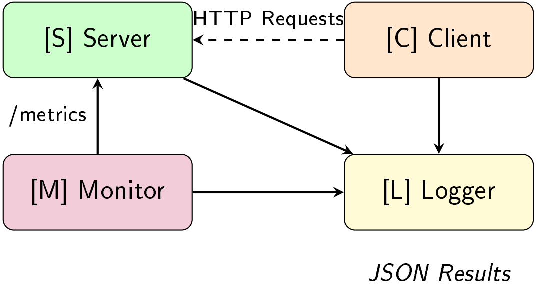
1. **Modular** – 4-component design
2. **Reproducible** – YAML recipes
3. **Scalable** – Native SLURM

Supported Workloads:

- vLLM Inference
- MinIO S3 Storage
- (Extensible to Vector DBs)



System Architecture: Four-Module Design



Server	**Client**	**Monitor**	**Logger**
vLLM, MinIO	Load Generator	Prometheus Scraper	Thread-safe JSON

Execution Flow: From Recipe to Results

Reproducibility: Declarative YAML Recipes

Recipe = Complete Experiment

- Model, container image
- Resource allocation
- Client concurrency
- Monitoring targets
- Cleanup actions

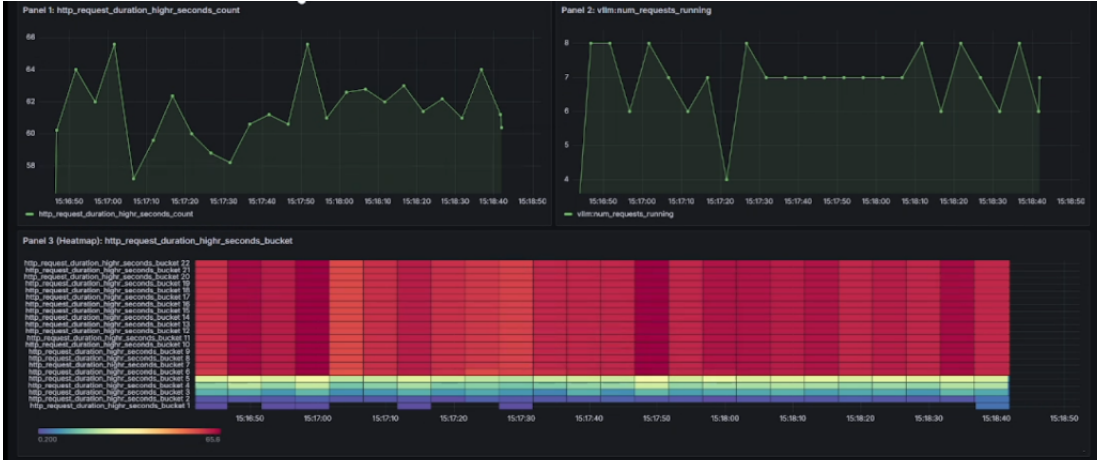
✓ **Git-versioned, shareable**

```yaml
services:
 - id: "vllm-01"
   executor: {type: process}
   command: |
     apptainer exec -nv ...
   healthcheck:
     url: "http://localhost:8000/health"
clients:
 - id: "loadgen"
   instances: 2
   workload: {type: vllm-inference}
```

Experimental Results: MeluXina GPU Partition

Metric	vLLM Inference	MinIO S3
Duration	120s	300s
Clients	2 instances × 4 threads	4 instances × 4 threads
Model/Target	facebook/opt-125m	Object PUT/GET
Total Ops	3,680 requests	∼6,800 objects
Throughput	15.2 req/s (per client)	145 MB/s (per client)
Latency Avg	125.5 ms	PUT: 530ms / GET: 360ms
Latency P95	195.7 ms	PUT: 960ms / GET: 670ms
Tokens/s	1,437 tokens/s	–
Success Rate	99.9%	100%

vLLM Inference: Real-Time Metrics



Panel 1: Request Rate | Panel 2: Concurrent Requests | Panel 3: Latency Heatmap

MinIO S3: Object Storage Performance

Panel 1	**Panel 2**	**Panel 3**
Total Requests (per client)	Traffic Sent (400-700 MB bursts)	TTFB Heatmap (latency buckets)

Built-in Visualization: FastAPI + Grafana

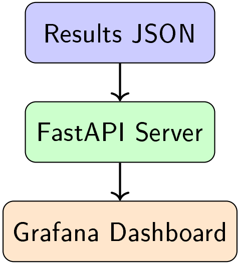
FastAPI Server Features:

- Grafana SimpleJSON datasource
- Auto-detects service type
- Applies `rate()` to counters
- Zero external dependencies

Usage:

`python fastapi_server.py`

→ Connect Grafana to `localhost:8000`



Extensibility: Adding New Workloads

Conclusion: Key Takeaways

What We Delivered:

HPC-Native	Reproducible	Validated
SLURM + Apptainer integration	YAML Recipe files	validated on MeluXina

Impact: Enables EuroHPC users to scientifically benchmark AI Factory components with minimal setup.

Questions?

Backup: Technology Stack

Component	Technology
Language	Python 3.9+
Configuration	PyYAML
HTTP	requests, FastAPI
Metrics	Prometheus format, matplotlib
Containers	Apptainer/Singularity
Scheduler	SLURM
Visualization	Grafana SimpleJSON

Backup: Error Handling

Multi-level Resilience:

- Healthcheck timeout $\rightarrow$ graceful shutdown
- Executor stop isolation $\rightarrow$ prevents cascading failures
- Workload error backoff $\rightarrow$ configurable retry with abort threshold
- Monitor resilience $\rightarrow$ continues on scrape failures