HPC Benchmark Toolkit

Challenge Evaluation Report

Team: 8 - EUMaster4HPC 2025/2026

Members: Alberto Finardi
Giovanni La Gioia
Laura Paxton
Giulia Lionetti

Platform: MeluXina HPC Cluster (LuxProvide)
Period: September 2025 – January 2026

February 2026

1 Introduction and Overview of the Work Delivered

1.1 Project Context

As part of the EUMaster4HPC challenge, our team was tasked with developing a tool for benchmarking Large Language Model (LLM) inference on High-Performance Computing infrastructure. The result is the **HPC Benchmark Toolkit**: a production-ready, open-source framework that automates the deployment, execution, monitoring, and evaluation of LLM inference benchmarks on the MeluXina supercomputer.

1.2 What We Built

The toolkit enables users to declaratively define benchmark experiments through YAML *recipes* and execute them on HPC clusters managed by Slurm with Apptainer containers. The system supports two inference backends - **Ollama** and **vLLM** (including multi-node distributed mode via Ray) - and is designed to be extensible to additional services.

At a high level, the framework consists of four major subsystems:

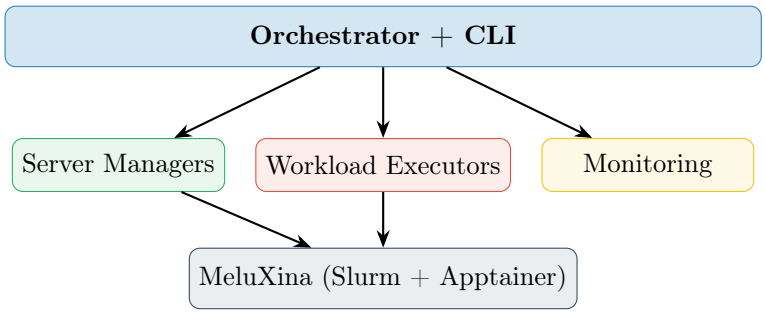

Figure 1: High-level architecture of the HPC Benchmark Toolkit.

Key capabilities include:

- **Recipe-driven experiments**: YAML configurations ensure full reproducibility, with support for parameter sweeps (Cartesian product over concurrency, batch sizes, prompt lengths).
- **Automated Slurm integration**: The CLI generates and submits SBATCH scripts, handles node allocation, and manages container lifecycles.
- **Real-time observability**: Prometheus metrics export and Grafana dashboards (accessed via SSH tunnels) for live throughput and latency visualization.
- **Distributed benchmarking**: Ray-based tensor parallelism across multiple GPU nodes for large-model vLLM deployments.
- **Comprehensive logging**: Distributed log aggregation from all nodes into structured JSON Lines format.

1.3 How We Evaluate the Results

We assess the quality of our deliverables along four dimensions:

1. **Functional completeness.** The toolkit successfully runs end-to-end benchmarks on MeluXina for both Ollama and vLLM (single-node and distributed). All seven execution phases - from recipe parsing to report generation - work as designed. Parameter sweeps produce the expected Cartesian product of trials.

2. **Metrics quality.** The system collects per-request latencies and computes p50/p90/p99 percentiles, throughput (requests/second), and error rates. Metrics are exported as CSV files for post-hoc analysis. A Prometheus/Grafana stack is also available for live monitoring, though its integration remains basic and would benefit from further refinement.
3. **Reproducibility.** Any experiment can be re-run by providing the same recipe YAML file. Recipes are self-contained and capture all parameters needed for a benchmark, preventing configuration drift across runs.
4. **Extensibility and code quality.** The codebase follows established design patterns (Factory, Template Method, Strategy). Adding a new inference service requires implementing three classes and registering them - no changes to the core orchestrator. The project includes API documentation, a developer guide, and a comprehensive README.

Limitations. The original project plan included additional deliverables such as an interactive web-based dashboard, advanced load patterns (Poisson arrivals, bursty traffic), and streaming inference benchmarking. These features were descoped during the project because the integration, debugging, and end-to-end testing effort concentrated on a "subset" of the team, leaving insufficient bandwidth to pursue all planned work items - see Table 1 and Table 2 for more details.

2 Work Organisation

2.1 Timeline and Work Calendar

The project spanned approximately four months (late September 2025 to late January 2026). Development was organised in four main phases:

Period		Activities
Sep 29 – Oct 10	*Phase 1: Design*	Requirements gathering and technical specification. Architecture design, recipe format definition (YAML schema), and initial repository setup. Distribution of responsibilities across modules.
Oct 10 – Nov 25	*Phase 2: Core dev.*	Core framework development: orchestrator, server managers, workload executors, and Slurm integration (Alberto). Logging subsystem (Giulia). Initial Prometheus/Grafana setup (Giovanni). Recipe format discussions (Laura). First end-to-end Ollama benchmarks on MeluXina.
Nov 25 – Dec 28	*Phase 3: Integration*	Integration of all subsystems into a working end-to-end pipeline. vLLM single-node and distributed (Ray) support. CLI development. Monitoring pipeline rework and dashboard refinement. Extensive testing and debugging on MeluXina.
Jan 2026	*Phase 4: Polish*	Live metrics pipeline finalisation. Documentation (README, API reference, developer guide, technical report, presentation). Final benchmark executions and validation.

Table 1: Project timeline and main activities per phase.

Table 2 summarises per-contributor statistics as reported by GitHub (`Insights` → `Contributors`, ordered by additions).

Contributor	Commits	Additions (++)	Deletions (−)
Alberto Finardi	13	19 722	3 841
Giovanni La Gioia	26	4 177	7 727
Giulia Lionetti	27	3 609	1 238
Laura Paxton	17	2 096	878
Total	**83**	**29 604**	**13 684**

Table 2: Git contribution statistics (source: GitHub Insights/Contributors).

2.2 Distribution of Tasks

Alberto designed the system architecture and implemented the orchestrator, all server managers (Ollama, vLLM, Ray cluster), all workload controllers and executors, the benchmarking logic, the CLI with interactive recipe creation, the Slurm sbatch generator, the distributed vLLM/Ray support, the monitoring metrics pipeline, and the project documentation (README, API reference, developer guide, technical report, presentation). He also served as integrator, coordinating module interfaces and performing end-to-end testing on MeluXina throughout the project.

Giulia designed and implemented the logging subsystem: the `BaseLogCollector` abstract interface, the `TailerLogCollector` with thread-based file tailing, multi-node log aggregation into structured JSON Lines, and the associated validation scripts.

Giovanni configured the initial Docker-based Prometheus and Grafana containers and created dashboard templates for Ollama and vLLM metrics.

Laura contributed to the initial requirements discussion and recipe format definition, and ran a few benchmark executions on MeluXina towards the end of the project.

Member	Primary Role	Key Deliverables
Alberto Finardi	Architect, Lead Dev. & Integrator	System architecture, orchestrator, server managers, workload controllers & executors, benchmarking logic, CLI, Slurm integration, distributed vLLM/Ray support, monitoring pipeline, documentation, overall integration
Giulia Lionetti	Logging	Logging architecture, `BaseLogCollector` interface, `TailerLogCollector` implementation, log aggregation, validation scripts
Giovanni La Gioia	Monitoring (initial)	Initial Prometheus/Grafana Docker setup and dashboard templates
Laura Paxton	Design & Testing	Participation in requirements, recipe format definition, benchmark runs

Table 3: Task distribution across team members.

2.3 Collaboration Tools and Workflow

We used GitHub (`EUMASTERHPC2526-TEAM-8`) with feature branches and pull requests. All members had MeluXina accounts under the same project allocation (`p200981`). Communication happened through online meetings and asynchronous messaging. Each module exposed a well-defined Python interface (abstract base classes), enabling parallel development with minimal merge conflicts; integration was performed incrementally.

3 What We Gained from the Challenge

3.1 Technical Skills

Working on this project provided hands-on experience with a full HPC software stack beyond what is covered in coursework:

- **HPC operations**: Slurm scheduling, Apptainer containers, multi-node allocation, and the constraints of shared supercomputer environments (queue times, limited GPU hours, no root access).
- **Distributed systems**: Ray-based distributed vLLM deployment, cluster coordination, cross-node network management, and debugging distributed failures.
- **Observability**: Building a metrics pipeline from in-process counters through Pushgateway, SSH tunnels, and Grafana dashboards.
- **Software architecture**: Designing a modular framework with Factory, Template Method, and Strategy patterns, and maintaining clean interfaces across independently developed subsystems.
- **LLM inference**: Performance characteristics of Ollama vs. vLLM, tensor parallelism impact, and the gap between theoretical and real-world GPU throughput.

3.2 Reflections

The most valuable outcome is the confidence from having built a non-trivial system end-to-end on real HPC infrastructure. Bridging the gap between tutorial examples and a multi-node benchmarking framework on a production supercomputer required solving many unglamorous but important problems: debugging SSH tunnels, handling container bind mounts, managing Slurm job lifecycles, and ensuring metrics flow from cluster nodes to local dashboards. If we were to start over, we would establish more frequent milestone check-ins and allocate more time for comparative benchmarks. The framework we delivered is functional, extensible, and documented - a solid foundation for future work.