PFE Projects Book

Email us at contact@redxt.com with your CV and a letter of motivation about the project you are interested in and we will get back to you.

Academic Year 2025/2026

ReDX

Project 1: HPC Software Stack Build Automation

Brief description:

In high-performance computing (HPC), efficiently managing and deploying software stacks across nodes and clusters is critical to achieving optimal performance. Manual software builds are time-consuming and prone to errors. Automating this process with robust tools ensures better reproducibility and higher productivity.

ReDX is working on improving a large scale cluster application software ecosystem on Intel Xeon CPUs and NVIDIA A100 and H100 GPUs. The primary goal of this internship is to automate the build of a software stack for this supercomputer environment. Then, the second goal is to streamline the maintenance of the software stack by automating the identification and cleanup of outdated software versions.

The student will gain practical experience in HPC build systems and automation techniques.

Current status:

The current cluster supports multiple compilers like gcc and intel compiler to ensure compatibility and performance for a variety of applications. A range of applications is also available to support research in multiple scientific domains, as an example, Quantum espresso that is used for electronic structure calculations and OpenFoam which is used for computational fluid dynamics (CFD) simulations. The cluster also offers several libraries like OpenMPI , LAPACK and FFTW and scientific frameworks, pre-compiled and managed through modules. These applications and libraries are installed using EasyBuild that automates software installation and management.

PFE project Goals:

The goal of the project is to build an HPC software stack for scientific computing and data science on an operational cluster. It is divided into **two key components**: the first focuses on automating the process using state of the art tools like easybuild or spack whenever possible. The second part focuses on parsing the modulefiles to retrieve the list of installed software, their versions, and compiler information, and dependencies, then generating a color-coded report to highlight which versions should be kept, removed, or reviewed.

Set up build automation:

- Use scripts, Spack, and EasyBuild to configure automated software stack deployment.
- Comparative study between these tools and the scripting method.
- Integrate the software modules to provide users with seamless access, to address the dependencies of the applications and to avoid conflicts between modules.
- Implement version control to track software changes (e.g., Git).

Software Stack Inventory and Filtering

- Automatically retrieve:
  - The list of available software installed on the Toubkal Supercomputer via modulefiles.
  - The version of each software.
  - The compiler version used for each software build (if available).
  - The dependencies on other modules

- Filter this inventory to keep only the last **N** versions of each software and remove the outdated ones by generating a color-coded report:
  - Red: Software versions to be deleted.
  - White: Software versions to be kept (**N** or less, if the corresponding software has less than N versions installed).
  - Orange: Versions with deprecated or problematic dependencies.

- To do that, Candidate should develop a script\software package that:
  - Parses modulefiles in the Supercomputer.
  - Takes as **input** the number **N** (how many versions to retain).
  - **Outputs** a filtered report.
  - Implement a "**restore mode**" that allows the recovery of previously deleted software versions, particularly those removed due to being outdated. This feature should enable users to selectively restore specific versions and regenerate the report to reflect how many software packages are being rescued from permanent deletion.

Required skills:

- Familiarity with Linux, and good software development skills with C/C++, Github, make/cmake, and other tools
- Ability to learn parallel programming models such as MPI, OpenMP, CUDA,...
- Knowledge and experience with Git, CI/CD tools is a plus.
- Very good English proficiency, well organized, use of project management tools like Clickup.

ReDX

Planned training:

- Linux fundamentals of the "Complete Linux Training Course" on Udemy platform, the course will be taken in modules starting with the fundamentals. The course has also advanced modules that can be taken during or even after the internship project during employment. A ReDX engineer will be with you along the way for any support required.
- Introduction to HPC: course with hands-on exercises to learn about HPC systems, and parallel programming using OpenMP, MPI and GPU programming with OpenACC.
- 1:1 sessions with ReDX engineer, and senior expert as required for the specific tasks of the project.

Duration and other details:

- Recommended period: 6 months.
- Compensation: Interns will have a monthly stipend with a possibility of an end of internship performance bonus as well as part or full time employment.
- Possibility to work with end customers.

ReDX

Project 2: LLM-driven Text Classification and Content Analysis

Brief Description:

This internship project focuses on developing a Large Language Model (LLM)-based framework for smart text understanding, categorization, and refinement. The goal is to extract structured insights from free-form text and generate improved, contextually consistent versions of the content. The system will enable accurate categorization, title suggestion, and adaptive refinement of textual inputs without manual intervention.

The project leverages pre-trained LLMs and few-shot learning techniques to perform text classification, metadata extraction, and semantic clustering, thereby supporting applications such as content organization, intelligent recommendations, and analytics.

Current Status:

Existing solutions often rely on rule-based or keyword-driven text processing, which lacks adaptability and semantic understanding. This project aims to automate text interpretation and refinement using LLMs, ensuring scalability, precision, and contextual awareness across varied input domains.

Project Goals:

1. Design an LLM-Based Text Understanding Module

- Fine-tune or prompt-tune a pre-trained LLM to interpret and structure textual content.
- Extract relevant metadata such as topic, intent, tone, or priority.
- Automatically generate concise and relevant titles and improved text versions.
- Ensure reliable classification and differentiation across multiple content categories.

2. Develop a Category Identification and Clustering Framework

- Implement a text-to-category mapping system for automated and dynamic categorization.
- Use semantic embeddings and clustering algorithms to group semantically related texts and discover new categories adaptively.
- Generate structured outputs suitable for integration with recommendation, ranking, or analytics subsystems.

3. Integration and Validation

- Integrate the developed models within an existing backend or prototype environment.

- Evaluate classification accuracy, refinement quality, and adaptability using diverse datasets.

Required Skills:

- Strong understanding of Natural Language Processing (NLP) and LLMs (e.g., GPT, T5, LLaMA).
- Experience in text classification, semantic embeddings, and few-shot learning.
- Proficiency in Python, with experience using AI libraries such as PyTorch, Transformers, and spaCy.
- Good analytical skills, strong English proficiency, and solid version control practices (Git).
- Knowledge of pruning techniques and hybrid web app development is a plus.

Deliverables:

- A technical report detailing system design, implementation, and evaluation.
- Source code and documentation for model integration and reproducibility.
- Performance metrics covering classification accuracy, text improvement quality, and clustering robustness.
- A demonstration prototype showcasing automated text categorization and refinement.

Duration and other details:

- Recommended period: 6 months.
- Compensation: Interns will receive a monthly stipend, with the possibility of an end-of-internship performance bonus, as well as the opportunity for part-time or full-time employment.
- Possibility to work with end customers.

ReDX

Project 3: Next-Gen Mobile Experience

Brief Description:

The current system is an AI-assisted platform for image collection; an internal pipeline ingests, organizes, and analyzes submissions to improve quality and fairness.

The framework is built on a hybrid technology where the web and Android versions are already in production. This internship focuses on delivering a production-ready iOS app, implementing a self-hosted update for Android (OTA updates without Play Store), adding an assistant chatbot to guide users and resolve basic issues, integrating a real-time user↔admin chat for advanced issues, and improving error handling/observability across mobile and web clients The project also includes a dual-app experience in one APK (two related sections under one install) to let users switch seamlessly between the system's core and complementary experiences.

Current Status:

- Web and Android (v1) are already in production. The iOS build exists, but it is outdated and missing feature parity.
- Updates for Android currently require users to uninstall/re-install APKs.
- Basic error reporting.
- Getting help inside the app is limited, with no real-time conversation or proactive guidance.

Project Goals:

1. Finish and ship the iOS app

- Match Android features (login, events, submissions, media, notifications).
- Smooth testing and release process (stable builds, clear release notes).

2. Self-update Android app with standalone OTA

- Users see an update prompt and install without manual delete/reinstall.
- Simple changelog, optional “remind me later,” and clear progress feedback.

3. In-app assistant (smart guide)

- Answer “how do I…?” questions, summarize event highlights and deadlines with either hard-coded entries or off-the-shelf LLMs,
- Suggests next steps that improve a user’s chance of success,
- Hands off to a real-time chat with a human when unsure.

4. Real-time chat (user ↔ admin)

- Quick help during events, with notifications and basic attachments (e.g., screenshots).
- Friendly, minimal UI focused on speed and clarity.

5. Better error handling & reliability

- Clear user-friendly messages, simple fixes (“Try again”, “Contact support”),
- Basic health checks and usage insights for proactive error handling.

6. Dual-App Prototype in One APK *(“two apps, one experience”)*

- A single Android app that lets users switch between two experiences (e.g., “Core” and “Community”) without a second install.
- The switcher (tab or toggle) shares the basics: one login, one update flow, one notification center, etc.

Required Skills:

- Angular + Capacitor (Mobile), Django (Backend).
- Practical mobile app building (testing, packaging, and releasing).
- Product thinking: clear wording, tidy flows, and consistent visuals.
- Team habits: small PRs, Git merging & branching, versioning, and release handling (tags, changelogs)
- Good communication skills and strong English level for specs, release notes, and support interactions

Deliverables:

- A ready-to-ship iOS app matching Android features.
- A private Android update flow inside the app (no Play Store dependency).
- A smart in-app guide (FAQ/assistant) that helps users self-serve.
- Real-time chat feature with notifications.
- Dual-App prototype in one APK (tab/switch navigation, shared session, unified settings).
- A demo showing the full journey
- A well-documented and clean code with a technical report detailing system design, implementation, and evaluation.

Duration and Other Details:

- **Recommended Period:** 6 months
- **Compensation:** Monthly stipend with potential end-of-internship performance bonus

ReDX

Project 4: Conversational & Argumentative Chatbot for HPC Cluster Storage (PFS-GPT)

1. Context and Motivation

Modern HPC facilities rely on parallel file systems (PFS) and scalable storage stacks to feed compute clusters at petascale throughput and millions of metadata operations per second. Popular choices include GPFS/IBM Spectrum Scale, Lustre (OpenSFS/Whamcloud), BeeGFS (ThinkParQ), CephFS, and emerging DAOS for disaggregated NVMe. Administrators, researchers, and students continuously ask overlapping questions—from definitions and component roles, to design trade-offs, to tuning and capacity planning). Documentation is fragmented, vendor-biased, and jargon-heavy.

Goal: Build a domain-expert chatbot that can *discuss, explain, and argue* storage designs, operations, and best practices across PFS technologies—serving **beginners** (friendly guided answers) through **experts** (deep technical guidance with citations), while minimizing hallucinations via retrieval-augmented generation (RAG) and verifiable reasoning.

2. Problem Statement

Create a multi user, citation-first HPC storage assistant able to:

- Answer **definitions**, **architecture** questions (e.g., client–MDS/OSS, OST/MDT layout, NSD/DAOS tiers), and **how-to** tasks (deploy, stripe, tune, troubleshoot),
- Provide **recommendations** driven by structured rules + retrieved evidence (workload patterns, capacity, bandwidth, metadata rates, cost),
- **Argue** pros/cons for design choices (e.g., Lustre vs BeeGFS for small-files, metadata scaling strategies) with clear, sourced reasoning,
- Adapt explanations to **Beginner / Intermediate / Expert** personas and export concise runbooks.

Constraints: Answers must be grounded in cited sources (vendor docs, standards, whitepapers, IO500 reports). The system must flag uncertainty and gracefully refuse dangerous advice

3. Tools and Resources:

- Hardware: 2–8 NVIDIA H100 GPUs, Lustre filesystem, Toubkal Supercomputer
- Software: HuggingFace Transformers, NVIDIA NeMo, FAISS, LangChain or LlamaIndex

ReDX

4.Learning Objectives

The intern will work in **close collaboration and mentorship with a PhD candidate pursuing world class publishable research** within an industrial and practical environment.

By the end, the student should be able to:

- Analyze the HPC memory landscape and its key variables.
- Design **data pipelines** that tolerate heterogeneous sources.
- Compare **alternative solution families** (rule-based, IR/semantic search, classical ML, LLMs, RAG, hybrids).
- Define **clear KPIs** (precision/recall of fields, reasoning correctness, robustness to change).
- Communicate design trade-offs and limitations honestly.

5. System Capabilities

- **Conversational QA (Beginner→Expert)** for GPFS/Lustre/BeeGFS/CephFS/DAOS.
- **Evidence-first answers** with citations and last-verified dates.
- **Design recommendations** (inputs → stripe policy, MDT/OST counts, tunables, HA notes).
- **Comparative argumentation** (pros/cons, assumptions, trade-offs across technologies).
- **Safety & ops**: guardrails/confirmations, telemetry, CI/CD, and exportable runbooks/configs.

6. Required Skills

- Python programming (HTTP requests, parsing HTML, basic data processing).
- Web data extraction basics (DOM parsing, tables, text).
- HPC basics
- Data normalization (units, naming, schema design).
- Basic database handling (DuckDB / PostgreSQL).
- Ability to document results clearly.
- **Intro AI/LLM skills (preferred):** prompting a Large Language Model to interpret specs, map unknown field names, summarize technical text.

7. Duration and Other Details

- **Recommended Period:** 6 months
- **Compensation:** Monthly stipend with potential end-of-internship performance bonus and potential paper publication co-authorship

ReDX

Project 5: Open-Source HPC Networking Chatbot (FabricsGPT)

1. Context and Motivation

In High-Performance Computing (HPC), the network fabric is as critical as compute and memory for scaling performance, ensuring predictable latency, and maintaining energy efficiency and total cost of ownership (TCO). Modern HPC clusters increasingly rely on advanced interconnect technologies such as InfiniBand (HDR, NDR, XDR) and high-speed Ethernet (100/200/400/800 GbE with RoCEv2 and DCB) to support massive AI and simulation workloads. These fabrics involve complex parameters—link speeds, ASIC generations, routing algorithms, congestion management (PFC, ECN), and offload capabilities (SHARP, GPUDirect, RDMA)—that collectively determine cluster performance and reliability. However, knowledge about these networking technologies is scattered across vendor documentation (NVIDIA/Mellanox, Intel, Broadcom), standards bodies (IEEE, OpenFabrics Alliance), whitepapers, HPC site guides, and community forums with best practices in architecting HPC networks for large scale clusters. These sources change frequently in layout, terminology, and available data fields. Integrating consistent and up-to-date networking knowledge into ReDX's **Cluster Configurator** or other HPC design tools is therefore time-consuming and fragile if done manually and requires substantial expertise and experience.

Goal: Build an autonomous, domain-specialized, LLM-assisted networking design assistant (**GPTFabrics**) that continuously discovers, interprets, normalizes, and reasons over InfiniBand and Ethernet information—and exposes valid insights (topology design, bandwidth/latency estimation, congestion control, offload compatibility) to the Cluster Configurator.

2. Problem Statement

Build a system that can, to varying degrees and by any justified approach:

- **Discover** relevant networking information (InfiniBand, Ethernet, topologies, RDMA/RoCE support, congestion control) from heterogeneous sources without relying on fixed structures or vendor-specific assumptions.
- **Understand** what it finds by normalizing networking concepts such as link speed (Gb/s), port radix, ASIC generation, latency, routing scheme …
- **Support design reasoning** (e.g., "Is this 2-tier fat-tree non-blocking for 128 GPU nodes?", "What's the oversubscription ratio with 200 Gb/s downlinks and 400 Gb/s uplinks?")

- **Remain robust over time** despite evolving nomenclature, topologies, and hardware generations (HDR→NDR→XDR, 400→800 GbE).

3. Tools and Resources:

- **Hardware:** 2–8 × NVIDIA H100 GPUs, Lustre filesystem, HPC Cluster
- **Software:** HuggingFace Transformers, NVIDIA NeMo, FAISS, LangChain or LlamaIndex
- **Data Sources:** NVIDIA/Mellanox docs, Intel and Broadcom Ethernet specs, IEEE papers, OpenFabrics Alliance resources, academic and industrial HPC network case studies.

4.Learning Objectives

The intern will work in **close collaboration and mentorship with a PhD candidate pursuing world class publishable research** within an industrial and practical environment.

By the end of the internship, the student should be able to:

- Analyze the **HPC fabric landscape**, differentiating InfiniBand and Ethernet in terms of latency, throughput, congestion behavior, and scalability.
- Design **data and retrieval pipelines** resilient to heterogeneous and evolving documentation formats.
- Compare **alternative reasoning strategies**: rule-based bandwidth calculators, semantic search, LLM prompting, fine-tuning, and hybrid RAG pipelines.
- Define **evaluation metrics** (precision/recall for field extraction, correctness of topology reasoning, hallucination rate, adaptability to new generations).
- Communicate clearly the **design trade-offs, limitations, and interpretability** of the system

5. System Capabilities

- **Find:**Locate pages, datasheets, or topology design guides that describe InfiniBand and Ethernet hardware (NICs, switches, optics, cables), congestion management techniques, and topology templates.
- **Extract/Map:**Identify and normalize key fields: link speed (Gb/s), port count, ASIC generation, latency, buffer depth, PFC/ECN support, RDMA/RoCE capabilities, routing type, topology constraints, and oversubscription ratios.
- **Explain:**Generate concise explanations for networking concepts such as “lossless Ethernet,” “non-blocking fat-tree,” “PFC vs ECN,” “RDMA over Converged Ethernet,” or “SHARP in-network reduction.”
- **Estimate:** Perform topology-aware calculations (node scalability, bisection bandwidth, port utilization, oversubscription analysis) and flag likely configuration issues (e.g., asymmetric leaf/spine counts or mixed link speeds).
- **Track Change:** Detect new hardware generations or unfamiliar field names and flag them for human inspection instead of silent failures—ensuring long-term maintainability as technologies evolve.

The student may implement only a subset of these features deeply, provided the methodology and evaluation remain rigorous.

6. Required Skills

The intern is expected to have high level understanding of these concepts and capable to learn a variety of required skills for the project including but not limited to:

- **Python programming:** HTTP requests, HTML parsing, data processing.
- **Networking fundamentals:** InfiniBand, Ethernet, RDMA/RoCE, PFC/ECN, topology design (fat-tree, Dragonfly, leaf-spine).
- **Data normalization:** units, naming standards, schema definition (speed, radix, latency, congestion control).
- **Database management:** DuckDB / PostgreSQL for structured data storage and retrieval.
- **Documentation skills:** clear technical writing and result presentation.
- **Intro AI/LLM skills (preferred):** prompting models to interpret technical text, map unstructured fields into schemas, and produce factual summaries of HPC network concepts.

7. Duration and Other Details

- **Recommended Period:** 6 months
- **Compensation:** Monthly stipend with potential end-of-internship performance bonus and potential paper publication co-authorship

ReDX

Project 6: Dynamic RAG Dataset for Multi-Cloud HPC

Description

This internship focuses on teaching our model to think like a cloud architect for HPC. Today, our LLM understands HPC code, but it can't yet turn that code into a concrete hosting plan on AWS, GCP, or Azure—let alone a mixed, multi-cloud setup. The aim is to collect full, functional architecture examples (single-cloud and heterogeneous multi-cloud), capture all the moving parts (compute families, storage tiers, networks, schedulers, quotas/limits, regions) and their costs, and shape them into a clean, provider-agnostic schema. Alongside that dataset, we want to build a weekly auto-refreshed knowledge base that keeps SKUs, prices, and capabilities up to date so the model's answers are grounded in facts, not guesses.

Finally, there will be a fine-tuning stage of the LLM on that curated dataset so it can read an HPC repo profile, weigh trade-offs, recommend an end-to-end cloud architecture, and—when helpful—emit a diagram that makes the design easy to understand.

Current status:

The current status of the existing LLM is fine-tuned on HPC code repositories (it understands code), but it does not know clouds and cannot map HPC → cloud architecture yet.
The intern will work in close collaboration and mentorship with a PhD candidate pursuing world class publishable research within an industrial and practical environment.

PFE project Goals:

1. **Collect & Curate Deployable Architectures**
    - Build a dataset of **single-cloud and multi-cloud (heterogeneous)** examples across **AWS, GCP, Azure**; fill the common schema, add short rationales and **cost snapshots**, and (optionally) a small diagram per example.
2. **Provider Component Catalogs**
    - For each provider, collect all relevant components (compute, storage, network, scheduler) with the necessary specs, limits, regional availability, and pricing.
3. **Hint Parsing & Mapping to Cloud**
    - Parse the user hint (desired hardware/architecture/budget), validate it against the given codebase, and map requirements to best-fit cloud components per layer (compute, storage, network, HPC cluster choice) and rank providers for each part.
4. **End-to-End Architecture Synthesis (with RAG)**

    - Have the LLM assemble a complete hosting architecture for the event, grounded by RAG (retrieved provider facts), producing a clear, structured plan (and optional diagram).
5. **Auto-Refresh Mechanism for RAG**
    - Implement a scheduled refresh (e.g., weekly) to update SKUs, prices, limits, and docs across providers, keeping recommendations current and factual.
6. **Evaluation & Demonstration**
    - Evaluate on unseen repos for fit-to-needs, clarity, and cost awareness; deliver a live demo where the LLM outputs an accurate, end-to-end, cloud-agnostic (or multi-cloud) architecture for a given HPC codebase.

Required skills:

- **ML/NLP basics:** dataset design, prompt/response schemas, instruction fine-tuning.
- **Cloud literacy:** familiarity with AWS/GCP/Azure building blocks (instances/VMs, storage, regions, pricing).
- **Data tooling:** Python, JSON, simple ETL/versioning; basic vector search/RAG.
- **Good practices:** clear documentation, neat labeling, reproducibility with Git.
- **HPC basics:** Awareness of **GPU/CPU differences**, schedulers (K8s/Slurm/Batch), and why networking/storage matter for parallel jobs.
- **Software practices:** Git, code reviews, clear documentation, and an eye for user experience and safety (validations, guardrails).

Deliverables:

- **Cleaned, preprocessed, well-presented dataset** of **HPC cloud architectures** (both single-provider and **heterogeneous multi-cloud**), in our common schema, with short rationales and cost snapshots; (optional) diagram per example.
  Auto-refreshed RAG dataset (tunable periodicity) covering at least 3 providers with components, SKUs, limits, regional availability, and prices, plus a simple retrieval API.
  Fine-tuned LLM layers on your architecture dataset (config + checkpoints or LoRA adapters).
- **Final capability demo**: the LLM accurately recommends an end-to-end cloud hosting architecture for a given HPC codebase—detailed, cost-aware, fact-grounded, and (when useful) accompanied by a diagram.
- **Technical report**: schema, curation process, RAG refresh, fine-tuning setup, and evaluation results.

Duration and other details:

- Recommended period: 6 months.
- Compensation: Monthly stipend with potential end-of-internship performance bonus and potential paper publication co-authorship

ReDX

Project 7: Infrastructure as a Code Normalizer & Execution Orchestrator

Keywords: Multi-Cloud HPC with Monitoring, FinOps, Security & Cross-Cloud Comms

Description

Build a cloud-agnostic execution layer that takes an ArchitectureGraph (emitted by your LLM/planner or hand-crafted for tests) and materializes it into deployable, validated HPC environments on one cloud or across multiple clouds at once. It generates portable Terraform from the graph, then uses Jenkins pipelines and Ansible to configure cluster software, drivers, storage mounts , and network. It ships with strong monitoring (metrics, logs, traces) and a FinOps layer (cost simulation, tagging, budgets/alerts, spot/preemptible strategies, egress checks).

Current status:

The system can spin up a single Kubernetes-based HPC cluster on AWS using Terraform, but only when given a narrow, pre-defined set of inputs. It does not yet support Google Cloud or Azure, cannot mix components across providers (e.g., compute on AWS with storage on GCP), and lacks flexibility in job schedulers (mostly K8s, no Slurm/AWS Batch paths). Post-provision setup (drivers, MPI/NCCL, storage mounts) is limited, and there's only basic visibility into performance and cost—no unified monitoring dashboards, alerts, or budget controls.
The intern will work in close collaboration and mentorship with a PhD candidate pursuing world class publishable research within an industrial and practical environment.

PFE project Goals:

1) Interface & Contract Layer (what we accept as input)

- Define a simple, cloud-agnostic architecture description (the "plan" the LLM or a user provides).
- Capture a few essentials only: workload type (e.g., MPI, GPU training), constraints (region, budget), and preferred services (if any).
- Provide clear validation and human-readable errors when a plan is incomplete or infeasible.

2) Provisioning & IaC Layer (how we create resources)

- Translate the plan into a portable **Terraform** that can target **AWS, GCP, and Azure**.
- Support **single-cloud** and **mixed** deployments (e.g., compute on one provider, storage on another).
- Keep a small, curated library of reusable modules (compute, storage, networking) that share consistent inputs/outputs.

3) Configuration & Orchestration Layer (how we make it HPC-ready)

- After provision, automatically **configure** drivers, libraries, and the chosen **scheduler** (Slurm, Batch, or K8s).
- Standardize mounts for shared storage and object stores; verify access and throughput.
- Provide simple **smoke tests** (e.g., HPL/HPCG/mini-apps) to confirm the cluster is healthy.

4) Security & Isolation Layer (how we protect teams and data)

- **Strong tenant isolation** per competitor/team:
  - Separate **accounts/projects/subscriptions** or hard **namespaces** with strict **RBAC** & **network policies**.
  - **Per-team KMS keys** and **per-team buckets/filestores**; default **private subnets** and least-privilege IAM.
- **Secrets & identity:** OIDC-based workload identity; **Secrets Manager / Key Vault / Secret Manager** for all credentials.
- **Audit & compliance:** centralized audit logs, access trails, immutable logs for dispute resolution.

5) Inter-Provider Communication Layer (secure cross-cloud links)

- Secure data paths between providers:
  - Encrypted **site-to-site VPN** / Interconnect options when needed; otherwise signed, time-limited object access.
  - **TLS-only** endpoints, presigned URLs, short-lived tokens; explicit **egress counters** and warnings.
- Clear patterns for **compute on Cloud A, storage on Cloud B** with **policy gates** (latency/egress thresholds).

6) Observability Layer (monitoring)

- Ship with **standard dashboards** for compute, GPU, storage, and job health.
- Enable **alerts** on common issues (e.g., node down, GPU throttling, disk saturation).
- Centralize logs and basic traces so failures are easier to explain and fix.

7) FinOps & Governance Layer (how we control spend and risk)

Before deploy: cost estimation for each plan/variant; compare options when relevant.

During competition:

- **Auto-suspend/teardown idle workspaces** (per-team policies): detect idle signals (no queued/running jobs, low GPU/CPU utilization, inactive storage IOPS) → **scale to zero** nodes, snapshot volumes, optionally **archive** to cold storage; **auto-resume** on new job.
- **Schedules** for nightly scale-down; **TTL** for ephemeral assets.

Policy controls: mandatory **cost tags**, **per-team budgets/alerts**, safe default to **spot/preemptible** with fallback.

Guardrails: block or require explicit approval for high-egress routes, out-of-policy regions, or over-budget plans.

Required skills:

- **Cloud platforms (at least two):** Practical experience with **AWS, GCP, or Azure** (VPC/VNet basics, instances/VMs, storage options, IAM).
- **Infrastructure as Code & DevOps:** Comfortable with **Terraform** (modules and variables), plus a CI tool (**Jenkins** or similar) and a config tool (**Ansible** or similar).
- **HPC basics:** Awareness of **GPU/CPU differences**, schedulers (K8s/Slurm/Batch), and why **networking/storage** matter for parallel jobs.
- **Observability & Cost awareness:** Familiar with **metrics/logging dashboards** and **cloud cost concepts** (on-demand vs spot/preemptible, egress).
- **Software practices:** Git, code reviews, clear documentation, and an eye for user experience and safety (validations, guardrails).

Deliverables:

- A technical report detailing system design, implementation, and evaluation.
- **Architecture Intake & Validation Interface:** A functional interface that receives the LLM's recommended architecture report, validates it, and translates it into a clear schema the IaC normalizer understands—plus generates a cost-estimation summary.
- **IaC Normalizer & Execution:** A working normalizer that takes the schema and automatically deploys/hosts the HPC competition according to the given single- or multi-cloud architecture.
- **Event Dashboard & Notifications:** A well-defined dashboard showing what's happening during the event (health, usage, progress, costs) with real-time alert/notification capabilities.

Duration and other details:

- Recommended period: 6 months.
- Compensation: Monthly stipend with potential end-of-internship performance bonus and potential paper publication co-authorship