PROCEEDINGS OF SCIENCE

An application-agnostic AI platform to accelerate Machine Learning adoption for basic to hard ML/DL scientific use cases

Mauro Gattari,[a,*] **Luca Giommi,**[b] **Marica Antonacci**[c] **and Gioacchino Vino**[c]

[a] *INFN LNF,*
Via Enrico Fermi 54, Frascati, Italy
[b] *INFN CNAF,*
Viale Berti Pichat 6/2, Bologna, Italy
[c] *INFN Sezione di Bari,*
Via Giovanni Amendola 173, Bari, Italy
E-mail: mauro.gattari@infn.it, luca.giommi@cnaf.infn.it, marica.antonacci@ba.infn.it, gioacchino.vino@ba.infn.it

Researchers at INFN (National Institute for Nuclear Physics) face challenges from basic to hard science use cases (e.g., big-data latest generation experiments) in many areas: High Energy Physics (HEP), Astrophysics, Quantum Computing, Genomics, etc. Machine Learning (ML) adoption is ubiquitous in these areas, requiring researchers to solve problems related to the specificity of applications (e.g., tailored models and intricate domain knowledge), but also requiring solving general infrastructure-level and ML-workflow related problems. As the demand for ML solutions continues to rise across the diverse research domains, there exists a critical need for an innovative approach to accelerate ML adoption.

In this regard we introduce an Artificial Intelligence (AI) platform designed as an application-agnostic Machine Learning as a Service (MLaaS) solution. The platform provides a paradigm shift by offering a flexible and generalized infrastructure that decouples the ML development process from the specific use cases. The AI platform is implemented as a software layer on top of our cloud platform: the INFN Cloud, a dedicated, geographically distributed infrastructure which offers composable, scalable, and open-source solutions. The AI platform leverages INFN Cloud resources and principles, gathering and orchestrating technologies to support end-to-end scalable ML solutions: Kubernetes, Kubeflow, KServe, KNative, Kueue, Horovod, etc., ensuring support for several ML frameworks: TensorFlow, PyTorch, Apache MXNet, XGBoost, etc.

This paper describes the platform's design and principles, as well as some selected use cases from Natural Language Processing and HEP domains that can take advantage of the "aaS" approach.

International Symposium on Grids and Clouds (ISGC2024)
24 -29 March, 2024
Academia Sinica Computing Centre (ASGC), Institute of Physics, Academia Sinica Taipei, Taiwan

*Speaker

PoS(ISGC2024)026

1. Introduction

At the National Institute for Nuclear Physics (INFN), scientists face challenges in many research areas: High Energy Physics (HEP), Astrophysics, Quantum Computing, Genomics, etc. INFN has five lines of research (Particle Physics, Astroparticle Physics, Nuclear Physics, Theoretical Physics and Technological Research) and facilities that extend throughout Italy, which participate in national and international projects and collaborations.

Across the diverse research domains, the demand for Machine Learning (ML) solutions has increased consistently. In order to accelerate ML adoption, we introduce an Artificial Intelligence (AI) platform: the platform is designed as an application-agnostic ML as a Service (MLaaS) solution, it is based on a flexible and generalized infrastructure and comprises a set of tools that allow to decouple issues related to the ML development process from the specific use cases. This is a novel approach to ML within INFN: physicists and computer scientists are collaborating closely together to collect, implement and orchestrate all required technologies that will accelerate the development of ML solutions for scientific use cases.

In the design of the platform, we followed a bottom-up approach: we started from the use cases. The idea is to address common use cases within our institute, collect reliable and consolidated technologies to solve these problems, then generalize the implementation to a set of services that all scientists can use for similar use cases.

In this paper we describe:

- the design and general ideas behind the AI Platform;
- the inference technologies and two use cases in different domains: Natural Language Processing (NLP) and HEP;
- the distributed training support offered by the platform.

2. AI platform

The general idea underlying the platform design is to provide services in the cloud that enable users and interfacing applications to:

- train/fine-tune ML models at scale;
- host/share datasets and trained models on the cloud;
- serve models to perform inference about new data;
- manage models and versions through a public INFN catalog;
- provide a Web Application with a proper UI/UX that hides to end-users the complexity of the platform;
- provide SDKs, CLIs and tools in general to accelerate integration with the platform.

The platform is based on an underlying layer: the INFN Cloud [1], our dedicated, geographically distributed infrastructure which provides computing, networking, and storage resources. On top of INFN Cloud we have several software layers that implement different functionalities. In Table 1, we summarize all the technologies collected to implement these layers and following we provide a brief description of the listed technologies.

Layer	**Resources/Functionalities**	**Technology**
Physical/Virtual resources	CPUs, GPUs, RAM, Storage, Networking	INFN Cloud [1]
Container Orchestrator	Automate deployment, scaling, and management of workloads on physical/virtual nodes	Kubernetes [2]
Event Source	Decouple task submissions from their execution	Kafka [3]
Inference	Models serving, horizontal scaling, batching, deployment strategies, inference pipelines	KServe [4], KNative [5]
Train	Run distributed training jobs, Hyperparameters tuning, Resources quota (multi-tenancy)	Kubeflow Training Operator [6], MPI Jobs [7, 8], Horovod [9], Katib[10], Kueue [11]
Infrastructure	Resources provisioning, Cluster Scaling	Kubernetes AutoScaler [12], INFN Cloud Orchestrator [13] and Infrastructure Manager (IM) [14]
NoSQL DB	Track jobs status, Datasets and Models catalog	MongoDB [15]
Object/Block Storage	Host data and models	S3/MinIO [16], Longhorn [17]

Table 1: AI platform layers

- **Kubernetes**: well-consolidated container orchestration technology;
- **Kafka**: high performance, scalable, streaming service, and message broker, that we use for decoupling execution from submission;
- **KServe** and **KNative**: technologies to orchestrate model serving for inference:
  - handle autoscaling, batching, deployment strategies (e.g., canary rollouts), inference pipelines;
  - KServe is multiframework: it enables serving models onto supported runtimes, e.g., TFServing, TorchServe, Triton Inference Server, etc.

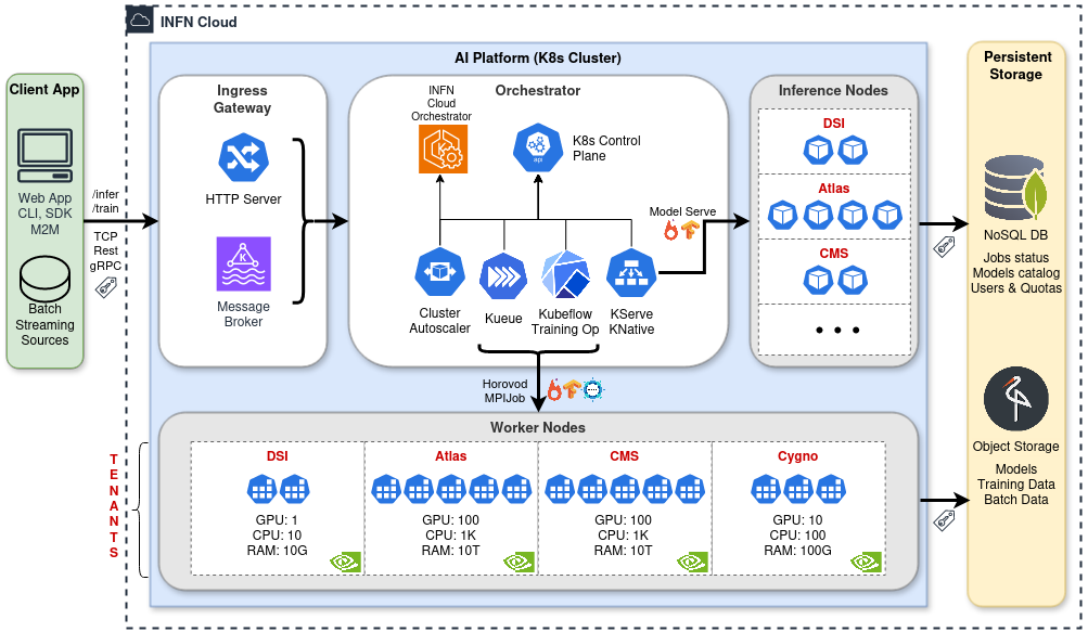

Figure 1: AI Platform: High-Level Architecture.

- **Training**: training jobs are resource-intensive and we need several technologies to orchestrate model training and resources management:
  - **Kueue** is a technology to manage resources quotas among tenants;
  - **Kubeflow Training Operator** allows deploying distributed training jobs and it is multiframework, i.e. it enables training on TensorFlow, PyTorch, MXNet, XGBoost, etc. (however, currently we only allow **MPI Jobs** (allreduce-style, implemented via Horovod), as they are the only ones that integrate with Kueue);
  - **Katib** is a software component for hyperparameters tuning.
- **Infrastructure**: Kubernetes AutoScaler, INFN Cloud Orchestrator, and Infrastructure Manager are technologies to orchestrate resources provisioning and cluster scaling;
- Finally, **storage solutions**:
  - **MongoDB**: high performance, scalable, NoSql database to flexibly store data;
  - **S3-compatible object storage**: stores models and training data with high availability and durability;
  - **Longhorn**: distributed block storage solution for Kubernetes that we use to fast access shared data within Kubernetes Pods.

2.1 High-Level Architecture

Figure 1 shows the high-level architecture of the AI platform where all components are in place and grouped as follows:

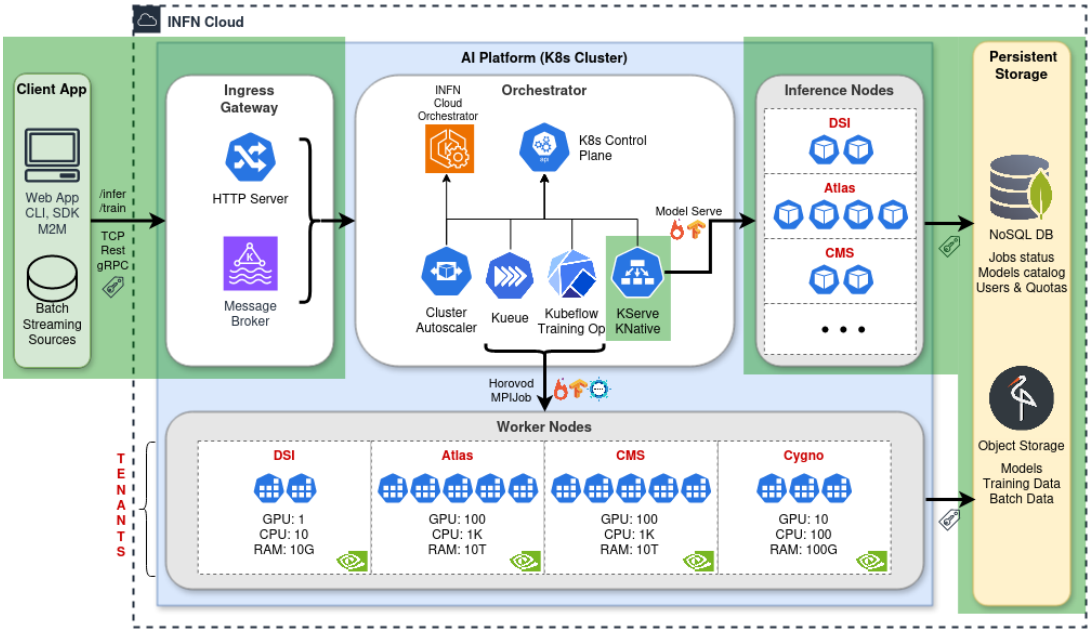

Figure 2: Inference: High-Level Architecture.

- **client applications**:
  - Web Apps, SDKs, CLIs and tools in general;
  - Machine-to-Machine (M2M) communication: application backends interfacing with the platform;
  - streaming sources: e.g., a running experiment that streams data on the fly to the platform for inference;
  - batching sources: e.g., at the end of an experiment run we collect data and deliver it in batches to the platform for inference.
- **ingress gateway**: this is the entry point to the platform:
  - HTTPs and gRPCs endpoints [18];
  - Kafka topics for data streaming and batching.
- **orchestrator**: coordinates the work of all components to provide the platform services;
- **inference nodes** for model serving and **worker nodes** for resource-intensive tasks, they are distinguished because they usually have different resources demand;
- finally, **persistent storage** technologies for storing models, data, metadata, etc.

2.2 Inference: High-Level Architecture

In Figure 2 we highlight (see green boxes) the platform technologies engaged in model serving. The main components involved within the orchestrator are KServe and KNative, these two

components manage model serving on a variety of frameworks: TFServing, TorchServe, Triton Inference Server, etc. They also handle autoscaling, deployment strategies (e.g., canary rollouts) and a variety of other features that give production-grade quality to the platform.

The following sections describe two use cases in different domains, NLP and HEP, that we implemented using the AI Platform inference technologies.

3. NLP use case: AI assistant

The first use case is in the NLP domain: we implemented and deployed through the AI Platform an AI Assistant, more precisely a Retrieval-Augmented Generation (RAG) pipeline (Figure 3).

AI Assistants can be used to solve a variety of tasks (e.g., searching for relevant information, data exploration and analysis, content creation, tasks automation, etc.) and nowadays Large Language Models (LLMs) are used to implement AI Assistants because of their compelling capabilities:

- LLMs offer "reasoning" capabilities;
- LLMs contain a lot of knowledge within their pretrained weights (so-called parametric knowledge) that can be surfaced by prompting the model.

However, LLMs also tend to hallucinate – i.e., generate false information – which indicates that the parametric knowledge they possess can be unreliable. RAG is a popular technique for injecting knowledge into a LLM, help solving hallucinations and a variety of problems:

- knowledge cutoffs (a pretrained model knows nothing after its pre-training date);
- poor understanding of specialized domains;
- save time/costs for LLM fine-tuning over proprietary data.

3.1 RAG pipeline

The workflow of a RAG pipeline is depicted in Figure 3 and at inference time prescribes three steps:

1. The user query is input to an **Embeddings Model**, which is a Language Model that outputs an **embedding**: n-dimensional dense vector that captures the semantics of the input text.
2. Perform a similarity search in a pre-indexed **Vector Database**: close vectors in the vector space correspond to semantically similar texts. Through semantic search we retrieve information related to the user question from the knowledge base. This relevant data is called **context**.
3. The context is then passed to the **LLM** along with the original query in an augmented **prompt** and an answer is produced.

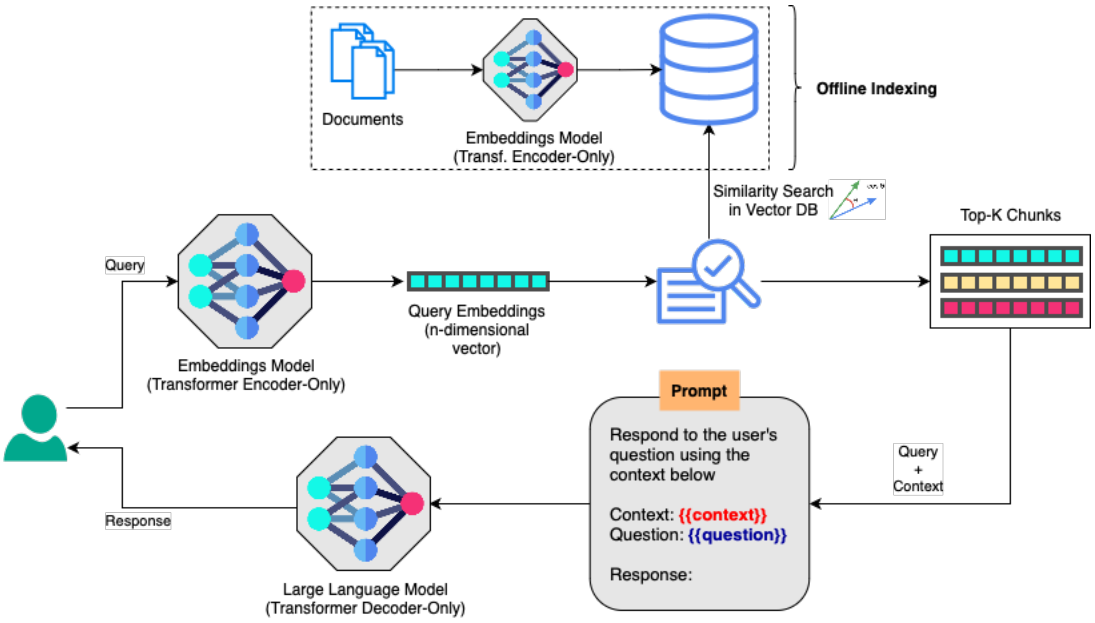

Figure 3: Overview of a RAG pipeline.

Basically, with RAG we augment the knowledge base of a LLM by inserting relevant context into the prompt and then we rely upon the LLMs ability to perform so-called "in-context" learning, i.e. the ability to leverage information within the prompt to produce a better output. In-context learning occurs at inference time during the forward pass (as opposed to pretraining). In addition, RAG exploits LLM "reasoning" (i.e., pattern recognition) capabilities and basic (i.e., parametric) world knowledge, which are used at inference time to rapidly adapt to or recognize the desired task.

As shown in Figure 3, in a RAG pipeline there are two kinds of Language Models, both are parts of the Transformers Architecture: Embeddings Models and LLMs, which we briefly describe in the following two paragraphs.

3.1.1 Embeddings Model

The Transformers Architecture was introduced in June 2017 in a paper named "Attention Is All You Need" [19] by researchers from Google Brain. The original architecture was composed of two blocks:

- an **Encoder**, designed to acquire understanding from the input text;
- a **Decoder**, designed for generating text.

Each of these parts can be used independently, according to the task:

- Encoder-only models (aka auto-encoders): good for tasks that require understanding of the input, such as sentence classification and named entity recognition;
- Decoder-only models (autoregressive models): good for generative tasks such as text generation.

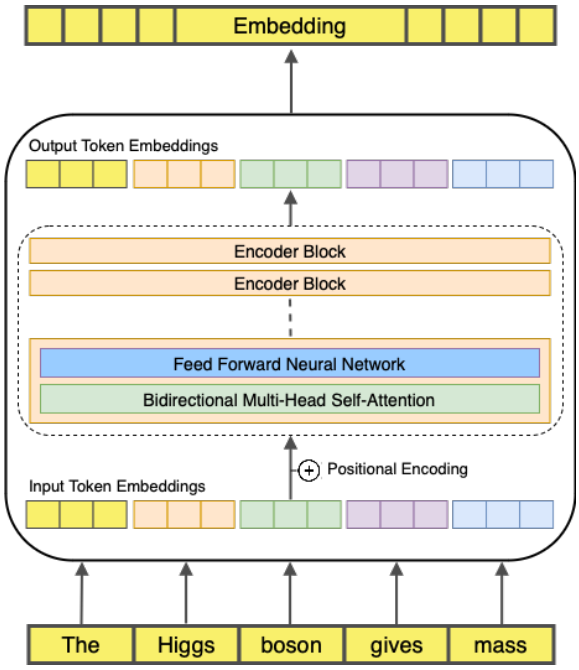

Figure 4: Encoder-Only Architecture.

Encoder-only models used for dense retrieval are usually bi-encoders, i.e. they are used to encode separately (potentially using different models) the query and the document (or passage) into a vector space. Bi-encoders are different from cross-encoders, which ingest both a query and a document using the same model and output a similarity score. Cross-encoders can more accurately predict textual similarity relative to bi-encoders, however, searching for similar documents with a cross-encoder is much more computationally expensive as it requires each textual pair to be passed through the model to receive a score. Notice also that recent research has indicated that decoder-only models can produce high-quality embeddings as well [20].

Figure 4 provides a sketch of the Encoder-Only architecture that takes an input text ("The Higgs boson gives mass") and outputs an embedding. If the Encoder has been properly trained, close vectors in the n-dimensional space correspond to semantically similar texts.

Currently, in our pipeline we are using an encoder-only model named *multilingual-e5-large-instruct* [21], a 559M parameters model with good Italian support (the model supports 94 languages).

3.1.2 Large Language Models

Figure 5 provides a sketch of the Decoder-Only architecture that takes an input text ("The Higgs boson gives") and outputs a probability distribution over next token (where "mass" corresponds to the token with highest probability).

Notice that the Decoder-Only architecture used by most modern LLMs is nearly identical to that of the original GPT model. We are just making models much larger, modifying them slightly, and using a more extensive training (and alignment) process.

Currently, in our pipeline we are using a decoder-only model named *Mixtral-8x7B* [22], a 46.7B parameters model with good Italian support (the model supports 5 languages).

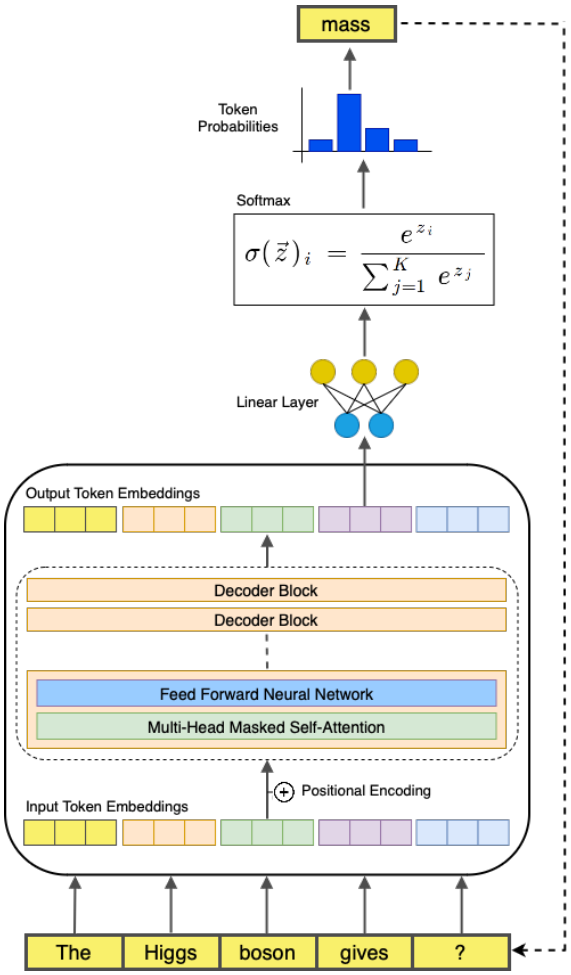

Figure 5: Decoder-Only Architecture.

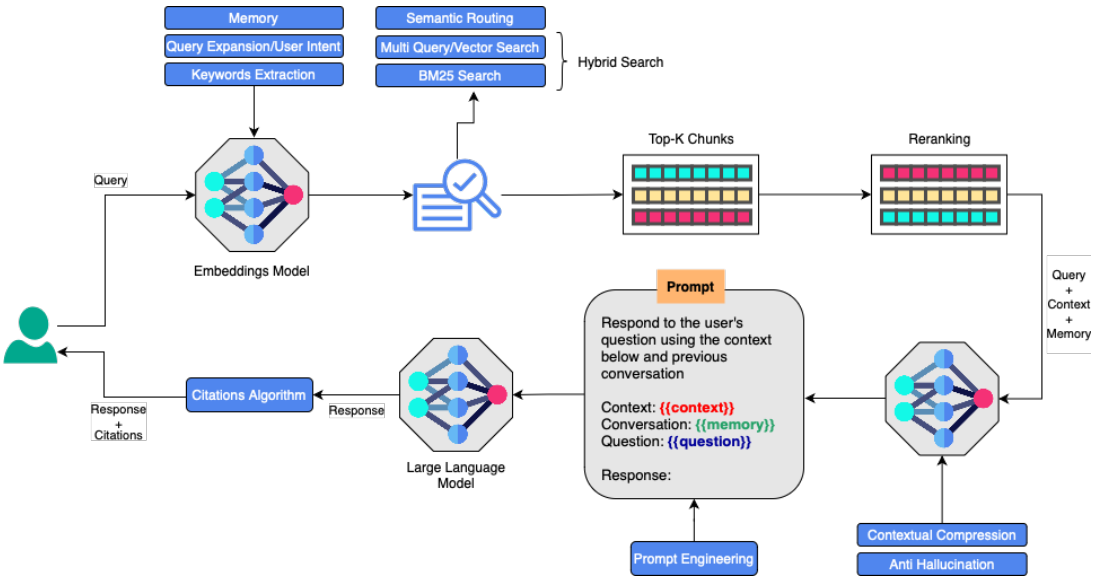

Figure 6: Full RAG pipeline.

3.2 Full RAG pipeline

A production-grade RAG pipeline can be quite complex (see an overview in Figure 6), as there can be so many details to address. For instance, we want to implement Memory, so that we can input queries that refer to previous passages in the conversation. Moreover, what happens when we input a follow up query like "give me more details", or "tell me more"? This piece of text cannot be searched in the vector database. To address this case we can use a technique called Query Expansion that rephrases the user query in a consistent standalone query that preserves the user intent and can be searched in the vector database.

In the following paragraphs we highlight some techniques usually employed in a RAG Full Pipeline.

3.2.1 Query Alignment

There can be a misalignment between the input query the user formulates in the form of a few words or a short sentence and the indexed documents, which are often written in the form of longer sentences or paragraphs. Solutions:

- **MultiVector Search**: we build additional vector stores, e.g., by indexing no longer the chunks embeddings but the embeddings of hypothetical questions (generated by an LLM) about the chunks.
- **Query Expansion / MultiQuery Search**: information retrieval systems can be sensitive to phrasing and specific keywords. We feed an LLM the user query (usually together with contextual data, e.g., previous conversion) and ask to rephrase the question in a single (Query Expansion) or a few (MultiQuery Search) consistent standalone query/queries. This technique preserves the user intent while improving the semantic search:
  - potentially match more semantic patterns through common synonyms and different paraphrases of the original query;
  - address cases where the user prompt is not that specific, think of a question like "give me more details".
- **Hypothetical Document Embedding (HyDe)** [23]: alternatively to (or in parallel with) Query Expansion, we can instruct an LLM to generate a hypothetical document for the given query and search for the latter. The hypothetical document is not real and may contain inaccuracies but may capture relevance patterns.
- **Semantic Routing**: sometimes we have multiple indexes for different domains and for different questions we want to query different subsets of these indexes. Semantic (or Query) routing is the process of classifying which index or subset of indexes a query should be performed on, this can be accomplished by using a small LLM for classification.

3.2.2 Hybrid search

Dense retrieval helps retrieve semantically relevant chunks for a given query, but it occasionally lacks precision in matching specific keywords. Depending on the use case, an exact match can be

necessary. There are techniques to extract keywords from the query (e.g., based on Embeddings Models or LLMs), then we can use lexical (i.e. sparse) retrieval (which relies upon a data structure called an "inverted index") to perform efficient keyword matching: given a bag-of-words representation of a query and a set of documents, the primary algorithm used for performing sparse retrieval is the BM25 [24] ranking algorithm.

With a hybrid approach, we can still match relevant keywords while maintaining control over the query intent.

3.2.3 Re-Ranking

The top K results are not necessarily ordered in the most relevant way. Re-ranking handles relocating the most pertinent information towards the edges of the prompt (LLMs tend to pay less attention to text in the middle of the context window [25]). Reordering can also be based on document diversity.

3.2.4 Contextual Compression

The most similar documents to the query in the embeddings space are not necessarily the most relevant ones nor the most semantically similar. In general, it is known that noise in the retrieved context adversely affects the accuracy of the answer generated by the LLM. This problem can be addressed by using small LLMs for contextual compression:

- filter out irrelevant, redundant, overlapping texts;
- compress less relevant contexts;
- highlight important paragraphs;
- overall reduce the context length.

3.2.5 Data Ingestion

Finally, the Data Ingestion step, through which the user's knowledge base is indexed in a vector database, is crucial to provide a consistent RAG pipeline. Here as well there are tools and techniques to improve the quality of this step:

- loaders: parse external data in different formats e.g., JSONs, PDFs, HTML pages, Doc files, etc.;
- data quality inspection and cleaning: clean up noisy documents (e.g., HTML tags), filter out confusing or redundant documents; do not combine e.g., FAQs that are inherently short with long PDFs, you'd better create two vector stores.
- splitters: chunk the raw data into smaller pieces of text;
- hyperparameters fine-tuning: experiment with different chunk sizes, chunk overlaps, top_k - e.g., a small chunk size might not catch enough context, while a big chunk size might introduce a lot of irrelevant noise.

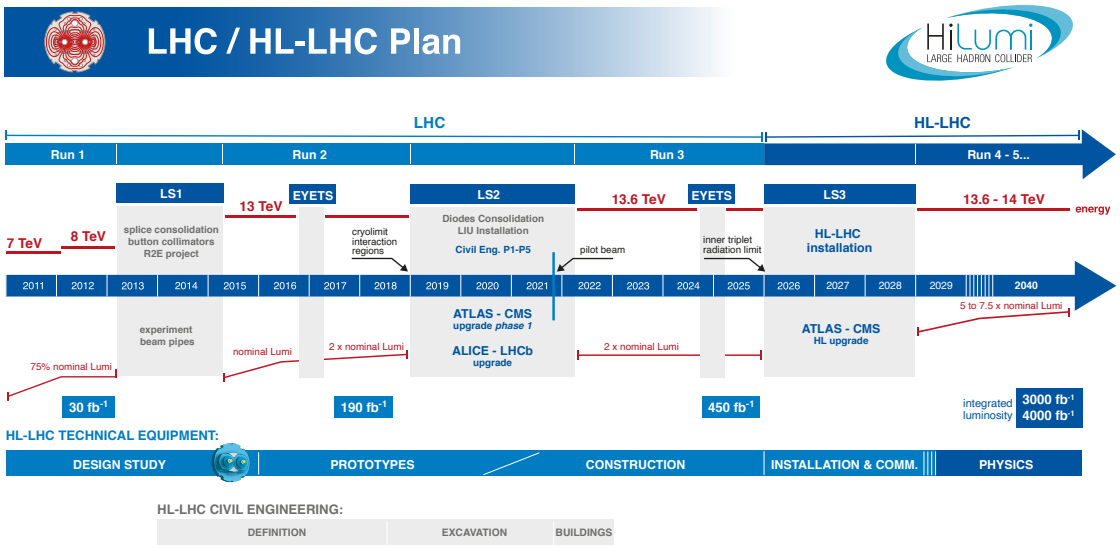

Figure 7: The current schedule for LHC and HL-LHC upgrade and run. Currently, the start of the HL-LHC run is foreseen at the beginning of 2029.

4. HEP use case

The Large Hadron Collider LHC [26] at CERN is the world's largest and most powerful particle accelerator, and it is the most important experimental apparatus that allows to advance the knowledge in the High Energy Physics (HEP) field. Collisions began to be produced at LHC from March 2010, and in July 2012 the discovery of the Higgs boson [27, 28], predicted by the Standard Model, was announced, representing a significant milestone in the history of LHC.

ML techniques are successfully used in many areas of HEP (online and offline reconstruction, particle identification, detector simulation, Monte Carlo generation, just to name a few) and there are many stories of success of their applications. For example they have been used in the analysis process that led the discovery of the Higgs Boson in 2012. ML is playing a crucial role in the LHC Run 3 and will also play it in the future High-Luminosity LHC upgrade. See Figure 7 for an overview of the schedule for LHC and HL-LHC upgrade and run.

Managing an entire ML project requires skills that generally physicists do not have. Thus it would be useful to create a service (a MLaaS) that easily allows to train a ML model and give predictions to be used in the analyses, hiding the underlying complexity to the physicists [29, 30]. MLaaS solutions are not yet widely used in HEP, although various R&D activities are underway within HEP aimed at providing HEP analysts with tools/services to accomplish ML tasks, but:

- either they target a specific step of the ML pipeline (e.g. inference phase) and do not cover the entire pipeline itself (from pre-processing, to training, and inference);
- or they are not "as a Service" (aaS) solutions;
- or they are difficult to generalize to other use cases.

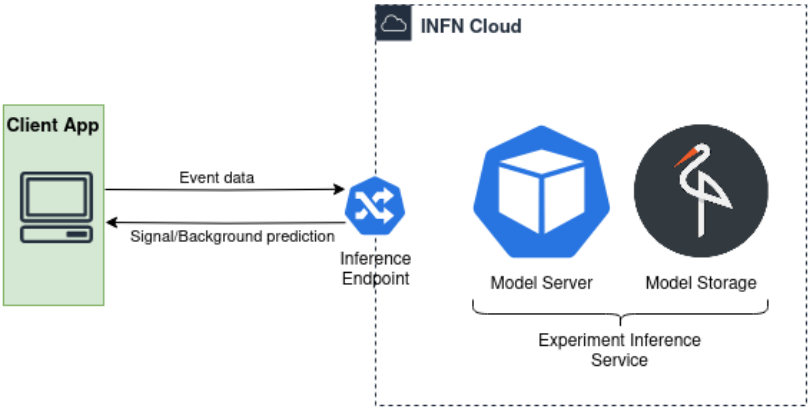

Figure 8: AI platform inference for the HEP use case.

At the same time many of the world's leading cloud providers offer different types of MLaaS services, including Amazon, Microsoft, Google, and IBM. These services offer pre-defined models that can be used to cover standard use cases, e.g. classification, regression, NLP, facial recognition, etc. But such solutions cannot be directly used in HEP:

- they cannot read HEP data directly in the ROOT data format (mostly used in the HEP world) and in most cases, ML deals with either CSV or NumPy arrays representing tabular data;
- generally HEP physicists apply pre-processing operations that are more complex than those offered by commercial providers.

The AI platform introduced in this paper has been designed to offer a MLaaS solution that is HEP-agnostic and in general application/experiment-agnostic: we presented in Section 3 a use case that leverages the inference technologies provided by the platform (section 2.2) to expose a RAG pipeline inference service; through the same technologies, we can serve a model for an inference use case in the HEP domain.

In particular, we used a ML model trained for a signal versus background discrimination problem in a $t\bar{t}H$ analysis in the boosted, all-hadronic final states. The model was hosted in the AI Platform object storage and served through KServe/KNative, see Figure 8. Through the AI Platform we could therefore provide an HTTP endpoint to make predictions for HEP collision events. Moreover, the scalable and batching features of KServe/KNative, together with the high performing streaming service Kafka, would allow to set up a pipeline for real time storage and inference of data produced by a running HEP experiment.

5. Distributed training

In Figure, 9 we highlight (see green boxes) the platform technologies engaged in distributed model training.

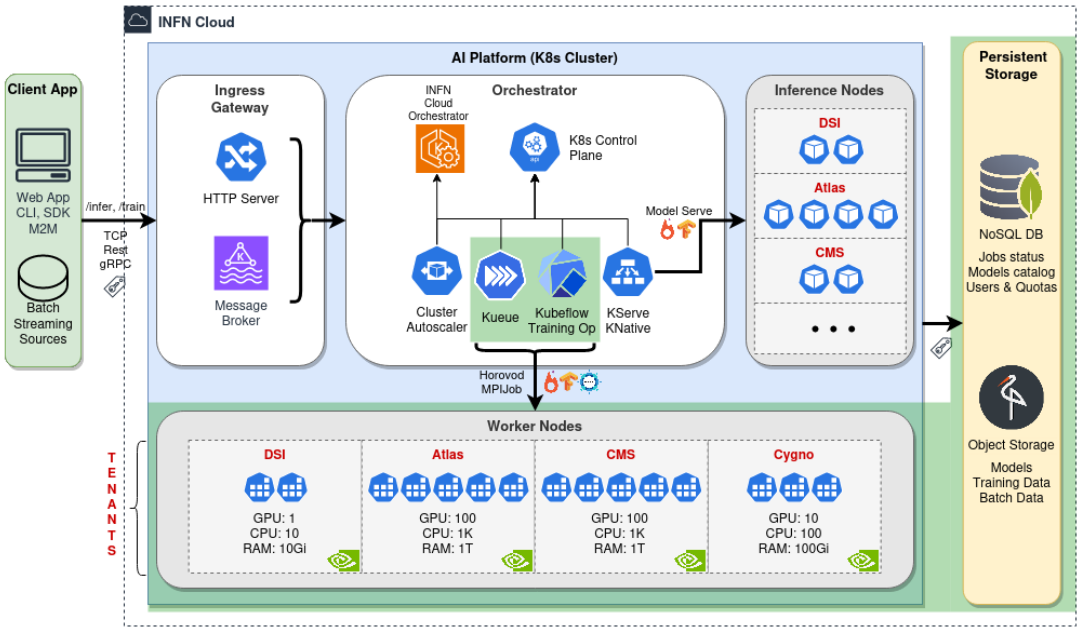

Figure 9: AI Platform Training Components.

The main component within the orchestrator that enables deploying distributed jobs is Kubeflow Training Operator [6]. Kubeflow Training Operator is a Kubernetes-native operator for fine-tuning and scalable distributed training of ML models created with various ML frameworks such as PyTorch, TensorFlow, XGBoost, and others.

Kubeflow Training Operator supports deploying MPI Jobs: users can run High Performance Computing (HPC) tasks with Training Operator and MPIJob since it supports running Message Passing Interface (MPI) on Kubernetes which is heavily used for HPC. Moreover, MPI Jobs deployed through the Training Operator integrates with Kueue [11]. Kueue is a Kubernetes-native system that manages quotas and how jobs consume them. Kueue decides when a job should wait, when a job should be admitted to start (as in pods can be created) and when a job should be preempted (as in active pods should be deleted).

The jobs deployment workflow is depicted in Figure 10 and prescribes the following steps:

1. a user submits a training job:
   - Kueue checks whether there are available resources to run the job within the tenant the user belongs to;
   - if there are not available resources, the job is added to a queue managed by Kueue;
   - if there are available resources, Kueue pass the job to the Training Operator for deployment: the job is run on several Worker Nodes (according to requested resources).
2. the job downloads model definition and training data from the Persistent Storage layer;
3. while training, checkpoints are pushed to the Persistent Storage layer.

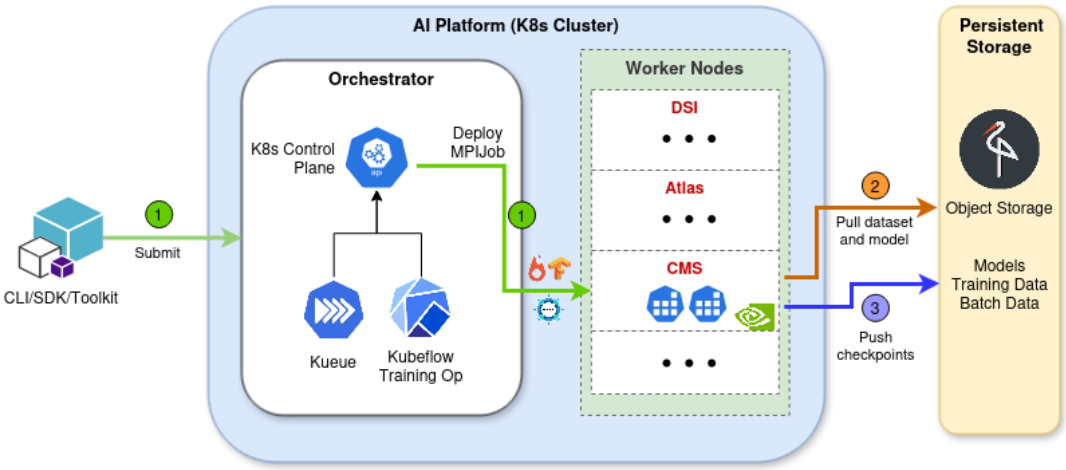

Figure 10: Jobs Deployment Workflow.

6. Conclusions

The platform's agnostic nature allows to address the practical challenges associated with deploying ML solutions in real hard science scenarios: streaming services, exabyte-scale storage solutions, high-bandwidth networking, support for native HEP data (e.g., CERN ROOT data format), etc. Furthermore, the AI platform promotes transfer learning and model reuse to accelerate the ML development lifecycle: users can leverage pre-trained models and share knowledge across different applications, reducing the time and resources required for training new models from scratch. This collaborative aspect not only enhances efficiency but also promotes a collective learning environment within the research community.

In conclusion, the application-agnostic AI platform serves as a unified ecosystem where developers, data scientists, and domain experts can collaborate seamlessly. By providing a standardized framework for ML model development, training, and deployment, the platform eliminates the need for extensive domain expertise in every application area. This democratization of ML empowers a broader audience to leverage the benefits of ML, breaking down barriers and fostering innovation across diverse research domains.

References

[1] *INFN Cloud. Cloud Services for Research*, https://www.cloud.infn.it

[2] *Kubernetes*, https://kubernetes.io/

[3] *Apache Kafka*, https://kafka.apache.org/

[4] *KServe*, https://kserve.github.io/website/latest/

[5] *KServe*, https://knative.dev/docs/

[6] *Kubeflow Training Operator*, https://www.kubeflow.org/docs/components/training/

[7] *MPI Jobs*, https://www.kubeflow.org/docs/components/training/mpi/

[8] *MPI Operator*, https://github.com/kubeflow/mpi-operator

[9] *Horovod*, https://horovod.ai/

[10] *Katib*, https://github.com/kubeflow/katib

[11] *Kueue*, https://kueue.sigs.k8s.io/

[12] *Kubernetes Autoscaler*, https://github.com/kubernetes/autoscaler

[13] *INDIGO PaaS Orchestrator*, https://github.com/indigo-paas/orchestrator

[14] *Infrastructure Manager*, https://github.com/grycap/im

[15] *MongoDB*, https://www.mongodb.com/

[16] *MinIO*, https://min.io/

[17] *Longhorn*, https://longhorn.io/

[18] *Open Inference Protocol*, https://github.com/grycap/im

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, *Attention is all you need* (2023), 1706.03762

[20] *A Practitioners Guide to Retrieval Augmented Generation (RAG)*, https://cameronrwolfe.substack.com/p/a-practitioners-guide-to-retrieval

[21] *Multilingual E5 Large*, https://huggingface.co/intfloat/multilingual-e5-large-instruct

[22] *Mixtral-8x7B*, https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

[23] L. Gao, X. Ma, J. Lin, J. Callan, *Precise zero-shot dense retrieval without relevance labels* (2022), 2212.10496

[24] S. Robertson, H. Zaragoza, Found. Trends Inf. Retr. **3**, 333–389 (2009)

[25] N.F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, P. Liang, *Lost in the middle: How language models use long contexts* (2023), 2307.03172

[26] O. Brüning, H. Burkhardt, S. Myers, Progress in Particle and Nuclear Physics **67**, 705 (2012)

[27] G. Aad, T. Abajyan, B. Abbott, J. Abdallah, S. Abdel Khalek, A. Abdelalim, O. Abdinov, R. Aben, B. Abi, M. Abolins et al., Physics Letters B **716**, 1 (2012)

[28] S. Chatrchyan, V. Khachatryan, A. Sirunyan, A. Tumasyan, W. Adam, E. Aguilo, T. Bergauer, M. Dragicevic, J. Erö, C. Fabjan et al., Physics Letters B **716**, 30 (2012)

[29] V. Kuznetsov, L. Giommi, D. Bonacorsi, Computing and Software for Big Science **5**, 17 (2021)

[30] L. Giommi, Ph.D. thesis, alma (2023), http://amsdottorato.unibo.it/10586/