eTRIKS Analytical Environment: A Modular High Performance Framework for Medical Data Analysis

Axel Oehmichen, Florian Guitton, Kai Sun, Jean Grizet, Thomas Heinis, Yike Guo
Data Science Institute, Imperial College London
{axelfrancois.oehmichen11, f.guitton, k.sun, jean.grizet16, t.heinis, y.guo}@imperial.ac.uk

***Abstract*—Translational research is quickly becoming a science driven by big data. Improving patient care, developing personalized therapies and new drugs depend increasingly on an organization's ability to rapidly and intelligently leverage complex molecular and clinical data from a variety of large-scale partner and public sources. As analysing these large-scale datasets becomes computationally increasingly expensive, traditional analytical engines are struggling to provide a timely answer to the questions that biomedical scientists are asking. Designing such a framework is developing for a moving target as the very nature of biomedical research based on big data requires an environment capable of adapting quickly and efficiently in response to evolving questions. The resulting framework consequently must be scalable in face of large amounts of data, flexible, efficient and resilient to failure. In this paper we design the eTRIKS Analytical Environment (eAE), a scalable and modular framework for the efficient management and analysis of large scale medical data, in particular the massive amounts of data produced by high-throughput technologies. We particularly discuss how we design the eAE as a modular and efficient framework enabling us to add new components or replace old ones easily. We further elaborate on its use for a set of challenging big data use cases in medicine and drug discovery.**

***Keywords*-Data Analytics, Data Infrastructure, Bioinformatics**

I. Introduction - Medical Big Data

The analysis of massive, multiscale, multimodal datasets has become a major challenge to address to convert data into knowledge and achieve innovation. This is particularly true in biomedical research, where scientists are increasingly facing a data deluge resulting from the rapid advances of high-throughput technologies. The complexity, diversity, context richness and size of biomedical data, such as Next Generation Sequencing (NGS) data, 'Omics, and imaging data, demonstrate the limitations of current systems.

The data collected by the UK Biobank [1] and the UBIOPRED project [2] are an illustration of the variety of data researchers are leveraging. The Biobank dataset currently contains imaging data (NIfTI and DICOM), genetic data (genotypes, SNP + sample QC information and imputed genotypes, etc.), clinical data (questionnaires, quantitative and qualitative phenotypic traits, etc.), assays (saliva, blood and urine), and many others. As for UBIOPRED, the project has collected clincal data and a large spectrum of 'Omics data including transcriptomics, genomics, lipidomics, drugomics, proteomics, breathomics and microbiome data. These datasets are characterized by extensive diversity in size and complexity, ranging from the measurement of the abundance of a few dozen analytes in drugomics, urine samples, to over 50,000 probes in transcriptomics analysis, to millions of reads of short sequences in microbiome data, consisting of terabytes of data.

Each data type represents its own set of challenges and requirements, either because it is a high dimensional format (transcriptomics or microbiome data), highly interrelated — e.g., imaging data (brain MRI) where each point in the matrix has a correlation with its neighbours — or extremely large in size (the genetic data in UK Biobank is currently 5 TB large and growing). Historically, these datasets have been studied relatively independently on separate platforms and in a specific context of a project or a disease. The interdependent nature of those different types of data, however, requires their combined analysis in order to discover more complex biological mechanisms. Analysis of the combined datasets thus requires new sets of tools. Thus, collection, management, storage and analysis of biomedical big data consequently make the development of new methodologies necessary. The challenges of developing systems for analysing multi-modal medical data consequently are:

1) the massive amounts of data needed for analysis.
2) the associated need of a scalable infrastructure.
3) the quickly evolving needs of analytics, i.e., the need for new and different algorithms and tools for data processing, integration and analytics.

In the context of the European Translational Information & Knowledge Management Services (eTRIKS), we developed the eTRIKS Analytical Environment (eAE) in answer to the needs of analysing and exploring massive amounts of medical data. The eAE is a modular framework which enables the analysis of medical data at scale. Its modular architecture allows for the quick addition or replacement of analytics tools and modules with little overhead, thereby ensuring support of users as the data analytics needs and tools evolve. As we discuss further, the eAE is flexible enough to support a variety of use cases across the biomedical domain.

II. Medical Analytics Background

In order to understand or discover biological processes, and thanks to advances in other fields such as physics and chemistry, medical research has increased the size and variety of data collected as we described in the previous section. That abundance of data has driven medical research to leverage an extreme wealth of analytics to explore and analyse the data.

A. Causality

Epidemiological research usually seeks to identify if any causal relationship exists between the risk factor and the disease. A traditional example that illustrates the association

between risk factor and disease is the impact of smoking on lung cancer [3]. However, the story of that person who smoked throughout their lives and never suffered from cancer shows that epidemiological problems are not straightforward. There is an association at work but exposure is a necessary but not sufficient condition for disease. Another problem of establishing causality in epidemiology is the necessity to reject other possible explanations for the observed association. Confounding factors may arise as well, causing a spurious association between dependent variables and independent variables. For example, many people who smoke heavily have low intakes of vitamins [4]. The Bradford Hill criteria [5] have been used to strengthen the evidence of causality in these types of studies [6] statistical methods as well as sophisticated and specialized methods have been developed in R, Stata or SAS to conduct research and uncover new causing factors.

B. Testing

A statistical hypothesis is a hypothesis that is testable on the basis of observing a process that is modelled via a set of random variables [7]. Statistical analysis aims at providing statistical insights about the datasets for further research, without any prior statistical knowledge, by performing multiple statistical tests on a given data set. Statistical hypothesis testing is a key technique of both frequentist and Bayesian inference, although the two types of inference have notable differences. Statistical hypothesis tests define a procedure that controls (fixes) the probability of incorrectly deciding that a default position (null hypothesis) is incorrect. The procedure is based on how likely it would be for a set of observations to occur if the null hypothesis were true.

Testing has been of crucial importance early on in biomedical research. Poor and complex signals, curse of dimensionality, computational needs of Bayesian to name only a few of the problems that researchers have faced. Many techniques have been successfully applied to overcome these problems. Principal component analysis (PCA) is a frequently used signal separation technique to discover potential subgroups of the dataset [8]. It uses an orthogonal transformation to convert observations of correlated variables into linearly uncorrelated ones (i.e. principal components), the number of principal components is less than or equal to the smaller of the number of original variables or the number of observations thus effectively reducing the dimensionality.

C. Clustering

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups.

Clustering analysis can fall into two types taking different kinds of input: feature-based clustering and similarity-based clustering. Feature-based clustering takes a feature matrix as the input and is applicable to raw noisy datasets. Commonly, finite mixture models, such as Mixture of Gaussians Model and infinite mixture models, such as Dirichlet process mixture model are used [9]. The basic idea of using mixture model is first fitting the mixture model with data and then computing the posterior probability of the data point whether it belongs to a cluster. The similarity-based clustering method, on the other hand, requires a distance matrix as the input and facilitates the domain-specific similarity.

In Bioinformatics, the clustering methods can be used to group similar samples and also similar features. For example, a gene expression dataset collected from multiple patients can be represented by a matrix, in which rows can represent genes and columns represent patients. The resulting matrix can well exceed terabytes in the context of a proteomics analysis. Clustering by columns (patients) can find groups of patients resulting in a possible patient stratification or discovering a correlation between genes and conditions [10][11].

D. Time series

A time series is a series of data points indexed (or listed or graphed) ordered by time. Most commonly, a time series is a sequence taken at successive equally spaced points in time and is thus a sequence of discrete-time data. Biological processes are often dynamic, thus researchers must monitor their activity at multiple time points. The most abundant source of information regarding such dynamic activity is time-series gene expression data[12]. Not surprisingly, generating time-series expression data has become one of the most fundamental methods for querying biological processes that range from various responses during development to cyclic biological systems. Recent improvements in methods for measuring gene expression such as high-throughput RNA sequencing (RNA-seq) and the increased focus on clinical applications of genomics make expression studies more feasible and relevant.

E. Prediction

In statistics, prediction is a part of statistical inference. One particular approach to such inference is known as predictive inference, but the prediction can be undertaken within any of the several approaches to statistical inference [13]. Indeed, one possible description of statistics is that it provides a means of transferring knowledge about a sample of a population to the whole population, and to other related populations, which is not necessarily the same as prediction over time. When information is transferred across time, often to specific points in time, the process is known as forecasting. Forecasting usually requires time series methods, while prediction is often performed on cross-sectional data.

Statistical techniques used for prediction include regression analysis and its various sub-categories such as linear regression, generalized linear models (logistic regression, Poisson regression, Probit regression, etc.). In case of forecasting, autoregressive moving average models and vector autoregression models can be utilized.

Deep learning algorithms, in particular convolutional networks, have rapidly become a methodology of choice for analyzing medical images and predicting patient trajectories [14][15].

III. The eTRIKS Analytical Environment

Our goal in developing the eAE is to enable the scalable exploration of multi-modal medical data using a flexible and modular architecture. In the following we discuss its architecture and the components we used.

A. Architecture - Design Principles

We design the eTRIKS Analytical Environment to provide users with an analytics environment which (a) has a user-friendly frontend, (b) has endpoints which can be easily integrated into tools, (c) is modular and finally (d) is also scalable in support of analysing large amounts of data.

We accomplish this by designing a multi-layer architecture of loosely coupled components to provide as much flexibility as possible. The modularity of this framework enables adding new components (public or private) or replacing outdated ones with better performing or proprietary ones. The eAE can also scale to user needs: small computations are executed locally while bigger ones are executed on scale-out infrastructure (e.g., cloud) and on specialized hardware (e.g., GPUs).

At the top, interacting with users, is the *endpoints layer* which hosts the containers which either provide the UI to users or the interface to integrate it into third party tools. The *endpoints layer* also contains the infrastructure to run smaller computations locally. Interacting with the *endpoints layer* is the *caching layer* which caches analytics results to avoid recomputation of frequent analysis, thereby making analysis more efficient. If a computation still needs to be computed, the *scheduling layer* will take care of it and schedule in on the *computation layer*. The *computation layer* provides the capability for the distributed computation of analyses and thus enables the scalability of the environment. The *computation layer* executes the computation on scale-out infrastructure like a cluster or the cloud and on specialised hardware.

B. Components and Tools

In the following we discuss the components used as well as the implementation of each layer of the architecture and discuss how the layers of the architecture interact.

1) General Environment: The operating system used on both the physical machines, virtual machines and containers within this architecture is Ubuntu 16.04 LTS. This version provides both, the stability required throughout the life of the machines and the necessary support for a large spectrum of libraries and drivers. Other Linux distributions, e.g., Centos or Debian, can also be used.

2) Endpoints Layer: For data exploration, the eAE relies on two sets of tools: tranSMART and a modified version of Jupyter. Both support a large spectrum of researchers ranging from biologists with limited computing and technological knowledge to more advanced users who worry which parameter optimisation will yield the best results. On the one hand, tranSMART will focus more on the hypothesis generation through a set of available workflows with their associated custom made visualizations. Jupyter, on the other hand, offers more possibilities as users can write their own scripts and visualizations to harness the power of libraries such as Matplotlib [16] or Lightning [17].

3) Caching Layer: eAE relies on two different tools for the caching layer: the NoSQL database MongoDB 3.2.5 in the eAE backend and SQL database Postgresql 9.3 [18] in tranSMART. MongoDB — which does not require a schema — is an excellent solution for adapting to any kind of data and acting as a cache. Another advantage is MongoDB's native support for high throughput read operations, scaling and resilience through sharding. PostgreSQL suffers from a number of limitations with respect to high availability and scaling. While solutions exist to tackle these limitations (pgpool [19] or pgclusterII), they are difficult to configure, set up and are not natively supported by PostgreSQL. Using MongoDB in place of PostgreSQL, on the other hand, results in a considerable speedup for the analyses as operations on the cache are substantially faster.

4) Scheduling Layer: For scheduling and monitoring of the clusters and jobs in the eAE, InterfaceEAE builds on OpenLava. OpenLava is an open-source, Platform Load Sharing Facility (LSF) compatible workload scheduler. InterfaceEAE identifies the requesting client to ensure the legitimacy of the request to perform access control. It also defines and implements the scheduling policies.

OpenLava is a workload management software designed to support the scheduling of high performance computing (HPC). We chose OpenLava as scheduling engine because it is an open source software application which makes it easier to optimise the implementation. OpenLava is capable of policy-based scheduling which can ensure fair allocation of resources. It supports Docker containers, cloud and VM resources alike.

The eAE uses a REST API that enables to: 1) remotely submit Spark, Python, GPU as well as R jobs; 2) monitor the health of the clusters; 3) monitor the jobs running. To achieve this, we developed a web application allowing us to create HTTP endpoints to the OpenLava CLI clients. This gives the capability to distribute jobs and subsequently activate them on the OpenLava cluster by communicating with the master node.

5) Computation Layer: The cloud platform chosen to support the on demand resources — to support scaling out when more compute-heavy or multiple computations need to be executed — is Openstack Liberty [20]. Openstack offers two particularly useful components to address this on demand requirement: the Heat[1] and Glance projects[2]. They enable creation of predefined templates or disk images to launch multiple composite cloud applications. Setting up the development environment therefore is seamless for the user and enforcing a version control of the software and libraries ensure the integrity of the environment. The eAE uses a private instance of OpenStack, however, there is no technical limitation to deploy it in public clouds such as Amazon AWS or Microsoft Azure. To further improve the architecture, we have replaced Heat and Glance by Docker containers which contain separate services to tailor a custom environment for the user and vastly improves performances for Jupyter. One user might, for example, require a specific set of languages (e.g., Python and R), while another user may need Spark support or GPU resources. Besides, we can support different versions of the software at the same time but hosted in different containers.

The eAE currently uses Cloudera CDH 5.9.0 for the deployment of the Hadoop [21] stack (including Spark [22]). We also considered MapR and Horton Works deployment tools. Each presents its own set of advantages and drawbacks. The reason for choosing Cloudera's is because it is arguably the best in terms of management interface and the availability of sup-

[1]https://wiki.openstack.org/wiki/Heat

[2]https://wiki.openstack.org/wiki/Glance

ported software in their stack. If the user's requirements were different, however, e.g., if the user prefers to use Amazon's AWS or own in-house components, some components would have to be adapted to adjust the interfaces of the eAE.

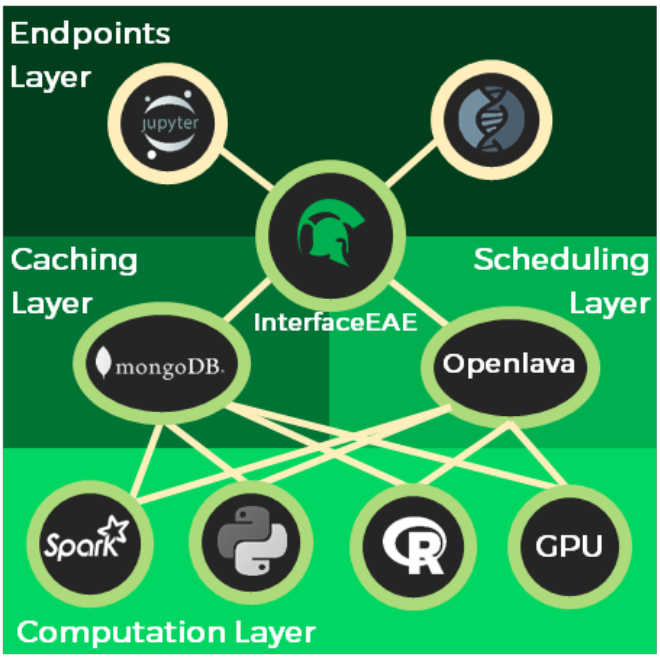

Fig. 1: **A schematic representation of the architecture of the eTRIKS Analytical Environment.**

6) Interaction between Layers: Figure 1 illustrates the architecture of the eTRIKS Analytical Environment. Each user owns a container hosting a modified version of the Jupyter server, a set of kernels (R, Python [23], Spark, etc.) and a set of standard libraries (Numpy, Scipy, scikit-learn [24], Bioconductor [25], etc.) supporting them. This server is one of the point of access of the eTRIKS Analytical Environment. Users can upload their data sets to the server and write their own scripts for analysis. Jupyter, through the selected kernel, sends the requested computations to the local engines which in turn sends it back to Jupyter. If the user requires more compute power, they can remotely submit their script to be scheduled on a larger centralized cluster to OpenLava. When the required resources become available, OpenLava triggers the computation. The Spark clusters are Hadoop stack production clusters installed on bare metals servers for performance reasons. Each one runs CDH 5.9.0 with the full Hadoop stack. The GPU clusters rely on TensorFlow 0.12 for Deep Learning and Nvidia CUDA 8[3] otherwise. The R servers rely on Microsoft R [26] Open [26], formerly known as Revolution R Open (RRO), which is Microsoft's enhanced distribution of R. The results are sent back to Jupyter or MongoDB (depending on the user's choice). The user can explore the results using advanced visualizations. The second entry point to the eAE is through a tranSMART plugin specifically developed for this integration. The plugin manages and interfaces with the MongoDB cache. The plugin can submit a job to OpenLava using data stored either in MongoDB or in tranSMART. The results are sent to the MongoDB cache. The user can explore the results in tranSMART and compare with previously run computations kept in their personal caches.

[3]https://developer.nvidia.com/cuda-zone

The table I summarizes the differences in the main features provided by comparable existing systems (proprietary or not): eAE, IBM Platform Conductor [27], Arvados [28] and Berkeley Open Infrastructure for Network Computing (BOINC) [29].

Platforms	eTRIKS Analytical Environment	IBM Platform Conductor	Arvados	Berkeley Open Infrastructure for Network Computing
Visualizations capabilities				
Jupyter	●	●		
Zeppelin		●		
Analysis support				
Spark	●	●		
Python	●	●	●	
R	●	●		
C/C++				●
Fortran				●
Go/Ruby/Perl			●	
Computation types				
CPU	●	●	●	●
GPU	●	●		●
Storage capabilities				
SQL	●	●		
NoSQL	●	●		
Content-Addressable Storage		●	●	
Monitoring & scheduling capabilities				
Jobs Status	●	●	●	●
Clusters Status	●	●	○	●
Complex batch processing	●	●		●
Workflow capabilities		●	●	
Interoperability				
REST API	●	●	●	
Distributed Clients	○	●	●	●
Platform support				
Installation procedures	●		●	●
Configuration documentation	●	●	●	●
Support available	●	●	●	●
Open Source project	●		●	●

● Fully supported
○ Partially supported

TABLE I: Feature comparisons between eAE, IBM Platform Conductor, Arvados and BOINC.

C. Data Exploration

Providing scalable data exploration in the eTRIKS Analytical Environment is accomplished using Jupyter notebooks.

The Jupyter notebook [30] is an open-source, interactive, language-agnostic HTML notebook application run as a web-based application. Jupyter offers more than 60 kernels, each

kernel providing a variable level of support for a specific language (e.g., Python or R) or software (e.g., Spark).

Jupyter currently does not fully support multiple users on the same instance. Jupyter Hub has partially solved the concurrency for some kernels (e.g. Python) but if multiple users run the same Spark kernel concurrently, their computation will have an undefined result or may crash. To overcome this limitation, we create pre-packaged images containing all the necessary kernels, libraries and dependencies required. Other than allowing for concurrent use of Jupyter, this strategy also allows users to work in a controlled environment. The use of standard images enables the transfer of the installation process from the user to a specialist. For simple installations like Python or R our strategy may introduce unnecessary overhead, but when facing complex installations and configurations such as Spark, it provides a substantial benefit. Regardless of the operating system (Windows, Mac OS or Linux) or the version the user's machine is running, the compatibility is ensured as the notebooks are running in a web browser. In addition, it also provides both consistency and flexibility. The results can be reproduced under the same conditions by sharing the notebook associated with the data. It enables flexibility as it relies on a virtual machine and can thus be scaled to any size required by the user. Furthermore, each Jupyter instance can support multiple kernels at the same time. Additional kernels or libraries can thus be added to customize the environment to address different needs.

Jupyter integrates code, markdown mark-up language and visualisation of the results in the same environment. This transforms scripts into story telling documents by incorporating comments, formulas and text to support the narration of the story behind the code. Thus, any slide presentations can be replaced by notebooks which can run simultaneously. Jupyter enables demonstrating the code by presenting the core of the code along with its outputs and also taking advantage of visualization libraries such as Lightning, Plotly or Matplotlib.

We developed a custom version of Jupyter as well to enable all users to remotely submit their workflows to the centralised clusters. Indeed, the cloud environment is designed to speed up the development process and provide data exploration and visualization functionalities, not intensive computing. This is why a scheduling and orchestration mechanism is required.

IV. Case Studies with the eTRIKS Analytical Environment

To illustrate how the eAE can be used for managing and analysing large scale translational research data, we implement several bioinformatics analysis pipelines, each primarily focusing on a different aspect of the eAE, scalability, user-friendliness, reproducibility and flexibility. The pipelines implemented include (a) iterative model generation and cross-validation pipeline for biomarker identification and (b) automatic sleep stage scoring based on raw single-channel EEG.

Each one is implemented similarly: the code is prototyped locally on a container for testing purposes while the computation on the full dataset is executed on clusters.

A. Iterative Model Generation and Cross-validation Pipeline

The iterative model generation and cross-validation pipeline implementation impressively illustrates the seamless scalability of the eAE. Using the eAE, the pipeline scales in the same rate as the underlying hardware, a crucial aspect given the massive amounts of data involved.

Cross-validation is a model validation technique for assessing how a statistical or computational model will generalise to an independent data set and it is thus used to estimate how good a predictive model is in practice.

In clinical trials, collecting further samples may be hazardous, costly or even impossible to do. In these cases, cross-validation is a powerful approach to prevent from testing hypotheses suggested by the data (called "Type III errors" [31]). To cross-validate, a model is first trained using a subset of the data set (i.e., training set) and the remainder of the data set is then used to test the model (i.e., testing set). To reduce variability, the data set can be partitioned into different training and testing sets and multiple rounds of cross-validation can be performed using them. Different statistical and computational approaches can be used for model generation including linear or non linear Support Vector Machine (SVM), Logistic Regression, Linear Regression, Alternating Least Squares, Lasso, etc. Non-parallelizable algorithms can be used as well.

These methods have been widely used in translational research, e.g., to identify the gene signature for stage II colon cancer [32]. Our implementation on top of the eAE enables us to scale up cross-validation by distributing the computation to multiple clusters which work independently to generate models. Each cluster randomly samples the training set and generates models using the selected algorithm. Each model is then tested against the test set to evaluate its fitness according to the specified set of indicators. By increasing the number of iterations, the model converges to the optimal solution.

Biomedical data always entails a large number of features, i.e., individual measurable properties that describe the observed phenomenon. To find the best fitting model, with the least amount of bias, we generate a family of models using the same dataset with different selections of features. Once the model is built, a certain amount of features can be removed, and a new model is generated using the remaining features. An unbiased approach is to randomly remove a selected number of features and then check whether it improves the fitness of the model. If the metrics remain the same then we move to the next iteration, if not then we select another set of features to remove. Removing a constant, small number of features at every recursion will avoid errors but is computationally expensive. A relative step, e.g., a percentage of the total number of features will speed up the process.

A more efficient approach is to introduce a very small amount of bias by lowering the chances of a feature, which we know beforehand is a factor, to be removed. By introducing this, the generation of models will naturally tend toward the optimal solution much faster. Another option, as suggested by Vladimir Vapnik's group [33] in the context of gene selection, is to use the weight magnitude as ranking criterion: compute the ranking criteria: $c_i = (w_i)^2$ for all i and find the feature with smallest ranking criterion $f = argmin(c)$.

The scoring used to assess the fitness of models can be done through a wide variety of measures, such as Area Under the Receiver Operating Characteristic (ROC) Curve - AUC, Sensitivity or True Positive Rate (TPR), Specificity or True Negative Rate (TNR), Negative predictive value (NPV), Positive predictive value (PPV) and F1-score. There is no metric that can give a correct answer for every situation. These metrics can only eliminate obvious "failures" due to performance, complexity, similarity to other models developed in similar ways or general stability. In the case of multiple thresholds with largely identical performance, the Hazard Ratio (HR) from Cox proportional hazards regression [34] can be used as a tiebreaker to favour higher HR values.

Once that first selection is done by each individual cluster, we gather them into a NoSQL cache where other services take over. The role of this section is to further narrow down the candidate models by comparing the models against one another. We then enrich the results to give as much information as possible to the scientist. It is thus important to implement different types of metrics to enable data scientists to select the right models for further extensive assessment or biological validation. Pathway enrichment is a method to identify classes of genes or proteins that are over-represented in a large set of genes or proteins, and may have an association with disease phenotype. Applying a pathway enrichment using the Kyoto Encyclopaedia of Genes and Genomes (KEGG) [35] and Gene Ontology (GO) [36] with multiple test corrections (Bonferroni, Holm-Bonferroni and/or FDR) on a sub selection of high scoring models can provide another insight to the results and an additional quality check for every model generated.

This type of unbiased approach to model generation is not well supported on standard platforms. The reason is that model generation is a long running computation and, the longer a computation runs, the more likely is a crash and loss of intermediate results. Indeed, the computations could be running for days at a time thus putting a lot of pressure on the hardware and software with no possibility to create milestones, a mechanism which would help to prevent full recomputation in the event of a crash. To address the issue of long-running computations crashing, we leverage the versioning mechanisms of Spark. The integration of Spark with the additional eAE layers on top enables users to run these large scale compute intensive experiments extremely easily and seamlessly through the end point of their choice. The stability, robustness and fault-tolerance of the platform enables these computations in a high performance fashion thanks to the fact that, even if a physical machine or a worker fails, the tasks get automatically rescheduled, thereby avoiding having to rerun the entire computation. The integration of Docker, Jupyter, Toree and eAE parts have enabled users to implement and prototype their algorithms efficiently without having to deal with how to set it up or link all the pieces together. Finally once the algorithm is ready, no modification is needed and the submission to the centralized cluster is straightforward.

B. DeepSleepNet

The implementation of the DeepSleepNet pipeline is an interesting illustration of the user-friendliness for highly parallelizable computation using the eAE. The eAE's user-friendliness allows users to quickly configure and launch the complex training of multiple models concurrently.

Sleep plays an important role in human health and being able to monitor how well people sleep has a significant impact on medical research and medical practice [37]. Sleep experts typically determine the quality of sleep using electrical activity recorded from sensors attached to different parts of the body. The entirety of the signals recorded with these sensors is called a polysomnogram (PSG). The PSG is segmented into 30s epochs, which are classified into different sleep stages according to standard manuals such as the Rechtschaffen and Kales [38] and the American Academy of Sleep Medicine. We develop a deep learning model (Python and GPU based), named DeepSleepNet, for automatic sleep stage scoring based on raw single-channel EEG. Our approach departs from the state of the art as we do not use any hand-crafted features such as time-frequency domain features [39].

One further constraint was that the two researchers were located on two different sites and were not computer scientists. To develop the new workflow, an eAE Jupyter container was deployed on a specific box with two GPU resources available to enable the two researchers to prototype their workflow simultaneously, share their code with one another seamlessly and access their data. To assess the quality of our model we use a 31-fold cross-validation. In each fold, we use recordings from 60 subjects to train the model and use the two remaining subjects to test the model. This process is repeated 31 times so that all recordings are tested. Finally we combine the predicted sleep stages from all folds and compute the performance metrics.

We run different iterations of the 31-fold cross-validation tasks on the eAE. Each cross-validation task takes roughly 6-7h to execute and the total execution time consequently is 170.5 hours. The only alternative is to either schedule the tasks on each machine of the cluster individually or sequentially on one machine. The former is tedious and far from practical as one needs to give the user access to all machines and the latter simply takes too long. The eAE provides a user-friendly web UI, which allows to train multiple models with different configurations concurrently across a cluster of high-performance machines. The scheduling of these tasks through the eAE takes approximately 2-3 minutes compared to an hour if done manually. Another benefit is the possibility to queue jobs to be run once machines become available. For this experiment, the GPU resources are only available at night as they are used for other projects during the day. The option to schedule two iterations at a time for 31-fold cross-validation tasks to be run during the night, without any external intervention, is a key feature for their timely delivery. In the case of this workflow, the experiments span an entire year.

V. Experimental Evaluation

A. Compute Scalability

We use benchmarks to evaluate the compute performance and scalability of the platform in terms of data size, number of executors and number of users. We execute the benchmarks on a cluster of six machines: one driver and five executors. All nodes are identical and each one has 200GB of disk

(Seagate, RAID 1 @ 7200RPM), 100GB of RAM (Micron, @ 1600 MHz), 24 cores (Intel Xeon, E5-2620 v2 @ 2.10GHz) and a network speed of 10GB/s. The cluster itself relies on Cloudera CDH 5.9.0 which comes with Spark 1.6.0. The Spark configurations used uses the yarn-client as master, 16 executor-cores, 20GB driver-memory and 20GB executor-memory.

We set the number of executors to five for the first compute scalability experiment but set it to different values in the second experiment. The analysis executed in the executor scalability experiment is a linear SVM which is readily available in Spark's machine learning library.

The experiments are designed based on an iterative model generation and a cross-validation pipeline. Our implementation on top of the eAE enables us to scale up cross-validation by distributing the computation to multiple clusters which work independently to generate models. Each cluster randomly samples the training set and starts generating models using the selected algorithm. Each model is then tested against the test set to evaluate its fitness according to the specified set of indicators. We use mRNA data from a GEO dataset (GSE31773) [40] as a base to synthesize the required data. The synthetic data is generated by first averaging the original values and then adding randomly generated noise for every new vector until the desired data size is reached. The label for each vector is randomly chosen. Six randomised datasets are generated independently, with sizes of 1MB, 10MB, 100MB, 1GB, 10GB, 100GB. The 1MB dataset is used to determine Spark's initialisation time. The initialisation time is subtracted from all other measurements in order to keep only the effective computation time of the model. The data is placed in HDFS for Spark to use. The results demonstrate the scalability of the platform as the processing time grows linear with the size of the data from 25.8s for 1GB to 290s for 100GB.

In the second experiment, we reuse the 100GB dataset generated previously but increase the number of executors to measure the impact on the execution time. The results in Figure 2 show a considerable decrease in the computation time when we increase the number of executors, thereby again demonstrating the scalability of the platform. The dip in execution time between 3 and 5 executors, however, shows that adding too many nodes can be detrimental to efficiency.

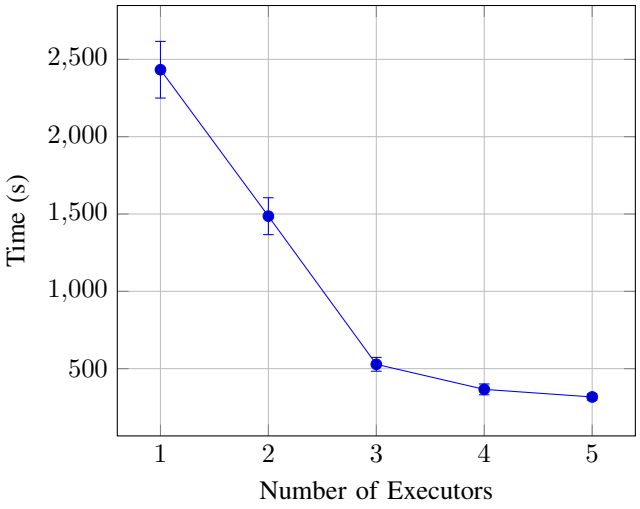

Fig. 2: The compute scalability of the eAE for increasing cluster size. Each point is the average running time of 30 experiments along with the standard deviation.

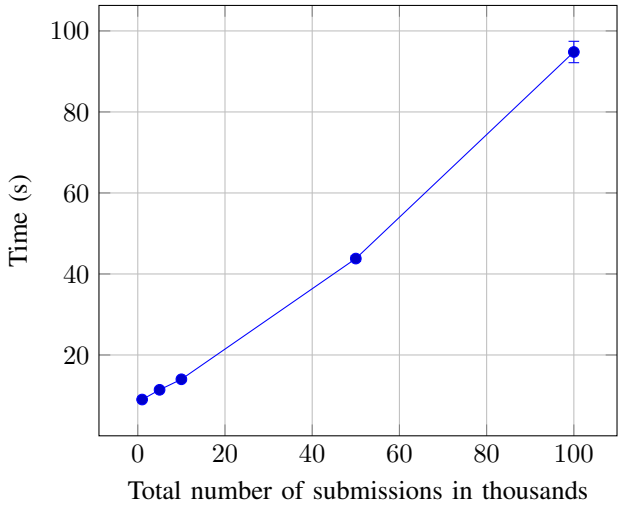

Fig. 3: The scheduling scalability of the eAE with respect to the submission size. Each point represents the average running time of 10 experiments along with the standard deviation.

B. Scheduling Scalability

To evaluate the orchestration scalability, we submit the same job concurrently an increasing number of times.

The experiments are based on a python demo script available as part of the eAE demo scripts[4] (including the fully dockerized version of the eAE). This example illustrates a basic linear regression coupled with a cross validation. The example only uses the first feature of the 'diabetes' dataset to illustrate a two-dimensional plot of this regression technique.

N independent clients (N=1,5,10,50,100 in this experiment) concurrently submit to OpenLava a batch of 1k jobs which are first scheduled, then computed and finally the results are returned to the user. Thus, the total number of jobs are submitted are 1k, 5k, 10k, 50k and 100k in this experiment.

[4]https://github.com/aoehmichen/eae-files/raw/master/jupyter-examples.zip

Figure 3 shows that submission and scheduling scales linear with a very large number of requests and is efficient even in a scenario of more than 10k which is rather unlikely in a production environment using the eAE.

C. User Scalability

The user scalability is still ongoing as the numbers of client applications is steadily growing. So far, there are ten Jupyters users actively using the eTRIKS Analytical Environment and one tranSMART demo server. There are no performance issues so far and the compute layer is efficiently shared across all users. Many of the users experience major speed-ups in their research as they are able to easily schedule N number of jobs to be computed during the night or running their cross validations concurrently across different nodes.

VI. Discussion and Conclusion

In this paper we discuss designing and using the eAE, an architecture for the efficient and scalable analysis of massive amounts of medical data. As we report, the resulting system provides an efficient and effective solution for data exploration and high performance bioinformatics. The use cases show that eAE's architecture proves flexible, scalable and robust for large scale data analysis in the context of translational research.

Although the proposed architecture is designed on open source software, it is possible to replace components with proprietary ones to improve performance. One of these components is OpenLava, which can be replaced by any other proprietary solution such as LSF. Another potential improvement is the integration of IBM's Spectrum Conductor with Spark to remove the need for local spark instances and to improve Jupyter's user experience and overall performance.

However, the current architecture presents two challenges. The eAE leverages different technologies and programming languages requiring a steep entry cost for new programmer to implement new clients or extensions. The second one is the way data is transferred across the different components, in the current form the interface manages the transfers through bash commands and scp connections. This limits the ability of the platform to accept data from remote clients and propagate the same data to different workers. Thus a dedicated component with HTTP based request for data transfer is being implemented to improve the flow of data between the client applications and the different components of the eAE. In order to reduce the technological stack as well some components will be written in NodeJS. One improvement we investigate is the integration of Apache Kafka in the architecture coupled with Spark to enable streaming processing.

Acknowledgements

The authors would like to thank the eTRIKS consortium and members of the Data Science Institute at Imperial College London, especially Cerys Morgan and Paul Agapow for their helpful comments. This work is supported by the Innovative Medicines Initiative (IMI) eTRIKS project and Guangdong Innovative Research Team Program (No.201001D0104726115).

References

[1] N. Allen, C. Sudlow, and et. al., "UK Biobank: Current status and what it means for epidemiology," *Health Policy and Technology*, 2012.

[2] D. E. Shaw, A. R. Sousa, S. J. Fowler, and et. al., "Clinical and inflammatory characteristics of the European U-BIOPRED adult severe asthma cohort," *European Respiratory Journal*, 2015.

[3] W. L. Watson and A. J. Conte, "Smoking and lung cancer," *Cancer*, 1954.

[4] A. F. Subar, L. C. Harlan, and M. E. Mattson, "Food and nutrient intake differences between smokers and non-smokers in the US." *American journal of public health*, 1990.

[5] A. B. HILL, "The environment and disease: association or causation?" *Proceedings of the Royal Society of Medicine*, 1965.

[6] F. Aghajafari, T. Nagulesapillai, and et. al., "Association between maternal serum 25-hydroxyvitamin D level and pregnancy and neonatal outcomes: systematic review and meta-analysis of observational studies," *BMJ*, 2013.

[7] A. Stuart and J. K. Ord, *Kendall's Advanced Theory of Statistics, Volume 1, Distribution Theory*, 1994.

[8] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2016.

[9] C. Bishop, *Pattern Recognition and Machine Learning*, 2006.

[10] Y. Kluger, R. Basri, J. T. Chang, and M. Gerstein, "Spectral biclustering of microarray data: Coclustering genes and conditions," 2003.

[11] A. Oghabian, S. Kilpinen, S. Hautaniemi, and E. Czeizler, "Biclustering methods: Biological relevance and application in gene expression analysis," *PLoS ONE*, 2014.

[12] Z. Bar-Joseph, A. Gitter, and I. Simon, "Studying and modelling dynamic biological processes using time-series gene expression data," *Nature Reviews Genetics*, 2012.

[13] D. Cox, *Principles of statistical inference.*, 2007.

[14] V. Gulshan, L. Peng, and et. al., "Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs," *JAMA: Journal of the American Medical Association*, 2016.

[15] A. Esteva, B. Kuprel, and et. al., "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, 2017.

[16] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science and Engineering*, 2007.

[17] "Lightning Viz," 2016.

[18] PostgreSQL, "PostgreSQL: The world's most advanced open source database," 2014.

[19] T. Pohanka, V. Pechanec, and M. Solanska, "Synchronization and replication of geodata in the ESRI platform," in *SGEM 2015*.

[20] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "OpenStack: Toward an Open-Source Solution for Cloud Computing," *International Journal of Computer Applications*, 2012.

[21] J. Dean and S. Ghemawat, "MapReduce," *Communications of the ACM*, 2008.

[22] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," 2010.

[23] G. V. Rossum and F. L. Drake, "Python Tutorial," *History*, 2010.

[24] F. Pedregosa, G. Varoquaux, and et. al., "Scikit-learn: Machine Learning in Python," *The Journal of Machine Learning Research*, 2011.

[25] R. Gentleman, V. Carey, and et. al., "Bioconductor: open software development for computational biology and bioinformatics," *Genome Biology*, 2004.

[26] R Core Team, "R: A Language and Environment for Statistical Computing," 2014.

[27] IBM, "IBM Spectrum Conductor."

[28] I. Curoverse, "Arvados — Open Source Big Data Processing and Bioinformatics."

[29] D. P. Anderson, "BOINC: A system for public-resource computing and storage," in *Proceedings - IEEE/ACM International Workshop on Grid Computing*, 2004.

[30] M. Ragan-Kelley, F. Perez, B. Granger, T. Kluyver, P. Ivanov, J. Frederic, and M. Bussonier, "The Jupyter/IPython architecture: a unified view of computational research, from interactive exploration to communication and publication." *AGU Fall Meeting Abstracts*, 2014.

[31] F. Mosteller, "A k-Sample Slippage Test for an Extreme Population," *The Annals of Mathematical Statistics*, 1948.

[32] R. D. Kennedy, M. Bylesjo, and et. al., "Development and independent validation of a prognostic assay for stage ii colon cancer using formalin-fixed paraffin-embedded tissue," *Journal of Clinical Oncology*.

[33] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene Selection for Cancer Classification using Support Vector Machines," *Machine Learning*, 2002.

[34] D. Cao, M. Hou, and et. al., "Expression of HIF-1alpha and VEGF in colorectal cancer: association with clinical outcomes and prognostic implications," *BMC Cancer*, 2009.

[35] M. Kanehisa and S. Goto, "KEGG: kyoto encyclopedia of genes and genomes." *Nucleic acids research*, 2000.

[36] M. Ashburner, C. A. Ball, and et. al., "Gene Ontology: tool for the unification of biology," *Nature Genetics*, 2000.

[37] K. Wulff, S. Gatti, J. G. Wettstein, and R. G. Foster, "Sleep and circadian rhythm disruption in psychiatric and neurodegenerative disease," *Nature Reviews Neuroscience*, 2010.

[38] J. Allan Hobson, "A manual of standardized terminology, techniques and scoring system for sleep stages of human subjects," *Electroencephalography and Clinical Neurophysiology*, 1969.

[39] K. Aboalayon, M. Faezipour, W. Almuhammadi, and S. Moslehpour, "Sleep Stage Classification Using EEG Signal Analysis: A Comprehensive Survey and New Investigation," *Entropy*, 2016.

[40] E. Tsitsiou, A. E. Williams, and et. al., "Transcriptome analysis shows activation of circulating CD8+ T cells in patients with severe asthma," *Journal of Allergy and Clinical Immunology*, 2012.