Understanding League of Legends through Machine Learning and Analysis of Complex Networks[1]

1st André Silva
Dept. Computer Science
FCUP
Porto, Portugal
andresilva@fc.up.pt
asilva@luxcorp.pt

2nd João Afecto
Dept. Computer Science
FCUP
Porto, Portugal
joao.afecto@fc.up.pt
afecto@luxcorp.pt

3rd João Antunes
Dept. Computer Science
FCUP
Porto, Portugal
joao.antunes@fc.up.pt

4th Pedro Vieira
Dept. Computer Science
FCUP
Porto, Portugal
pedrocvieira@fc.up.pt
pvieira@luxcorp.pt

+ ***Abstract*—In this work, we investigate the dynamics of the MOBA game League of Legends. We study the inter-connectivity created by the matchmaking system, the overall statistics of players and experiment on rank prediction. Furthermore, due to the streaming nature of the data, we built software and setup hardware capable of dealing with dynamic volumes of data.**

***Index Terms*—Cloud, Cluster, Redundancy, Distributed Storage, CPU computing, GPU computing, Machine Learning, Network, InfiniBand, High Availability, Large-Scale Network Analysis, Complex Networks**

I. Introduction

Online gaming is a trend that has been popular for many years and, with the development of powerful hardware with increasing availability to the public, it is expected that it so remains for many years to come. A popular type of game is Multiplayer Online Battle Arena (MOBA). Examples of games in such genre are Smite, Dota 2, Pokemon Unite and the object of study of this work, League of Legends. More details about the game will be presented when deemed necessary to understand the conclusions. Specifically, section V-C gives an overall description of the game.

The multiplayer online battle arena (MOBA) represents a subgenre within the realm of strategy video games, wherein two teams of players engage in competitive gameplay on a predetermined battlefield. Each participant assumes control over a single character possessing a unique array of abilities that are enhanced throughout the game, thereby influencing the strategic direction of their team [7].

As stated previously, our work will revolve around League of Legends (LoL). The developer of LoL, Riot Games offers ten distinct server regions [2] for players based on their location. We will work only with data from the region Europe West (EUW1). The data used was extracted using the official "Riot API" [1].

The main goal of our study is to better understand how LoL works, being that the distribution of players per rank and statistics related to player interaction with the game options (e.g. champion pick rate), the behaviour of the matchmaking system through understanding how the players connect, and study the possibility of rank prediction through a selected amount of features.

Since data has to be acquired from the Riot API, we needed to build software to pull said data. Moreover, the software used must possess the capability to proficiently query data in order to mitigate superfluous iterations of queries. Furthermore, the software must possess the capacity to store the data in a manner that facilitates seamless integration with the preferred search engine.

In terms of hardware, since we will be dealing with streaming data with undefined size *à priori*, we must have hardware that does not impose a limitation on the work being done.

As for software used for analysis, we have the possibility of either using the traditional Python approaches e.g. scikit-learn, networkx, pandas or experimenting with software built for large-scale analysis that takes advantage of CPU multi-core architectures and GPU compute capabilities. We will provide a simple comparative approach for the general machine learning algorithms. As for the analysis made for network analysis, we will only show the results of the analysis using large-scale analysis software due to the incapacity of the traditional software either by memory efficiency or by severe difference in computation time making said approaches infeasible at our scale of data.

II. Our own Cloud?

Once we decided to make our cloud, we defined the hardware that we would have available to each component to make it happen. After some short calculations we believe that the infrastructure we deploy is evaluated approximately in 400 000€. See acknowledgements (section VI).

A. Storage Infrastructure

1) Storage Hardware:

- 5 machines with:
  - 21x 10TB main storage HDDs
  - 3x NVMe 3.5TB journaling storage SSDs
  - 1x NVMe 350GB boot SSD
  - 1x Mellanox Connect-X 6 200Gb

[1]This work isn't endorsed by Riot Games and doesn't reflect the views or opinions of Riot Games or anyone officially involved in producing or managing Riot Games properties. Riot Games, and all associated properties are trademarks or registered trademarks of Riot Games, Inc.

Totalling over 1PB of storage (1050 TB raw storage).
The network fabric of this system is composed of two Mellanox QM8790 Quantum HDR Edge switches [13], which are Infiniband switches with 48x QSFP28 200 Gb ports. Each machine has 2x 100 Gb links active, one for the Ceph cluster network, another for the Ceph public network, where compute hardware can reach the cluster. Speaking of which...

2) Storage Software: Ceph [3] is a unified system for object, block and file storage, designed to be scalable, reliable and performant [4]. All these are qualities we need and want for the backing storage supporting all current and future compute requirements. The methods through which Ceph achieves it's impressive characteristics and feature set are beyond the scope of this article, but we highlight a few here: CRUSH [21], a pseudo-random non-cryptography hash function that serves as the backing for data distribution and retrieval, and it's integration into RADOS [22], the storage service that underpins the entirety of Ceph.

B. Processing Hardware

For the compute infrastructure, we had 7 machines with functionally equivalent hardware, as follows:

- 2 machines with:
  - 4x NVIDIA RTX A6000
  - 2x AMD EPYC 7443 24-Core 2.85GHz
  - 16x DDR4 3200MHz 32GB
  - 4x NVMe 1.7TB SSDs
  - 1x Mellanox Connect-X 6 200 Gb NIC
  - 1x Mellanox Connect-X 4 Lx 25 Gb NIC
- 5 machines with:
  - 4x NVIDIA GeForce RTX 3090
  - 2x AMD EPYC 7443 24-Core 2.85GHz
  - 16x DDR4 3200MHz 32GB
  - 4x NVMe 1.7TB SSDs
  - 1x Mellanox Connect-X 6 200 Gb NIC
  - 1x Mellanox Connect-X 4 Lx 25 Gb NIC

The two sets of machines are closer to computational power parity than it may seem to the uninitiated in this class of hardware. In fact, the RTX 3090 [19] and the RTX A6000 [20] include the same GA102 (Ampere) graphics processing core by NVIDIA. Though the active compute units in both products differ slightly, giving the A6000 with the theoretical performance victory over its sibling the 3090, by far its biggest upside is supporting 48 GB of GDDR6 VRAM, as opposed to the 3090's 24 GB of GDDR6X VRAM. While the latter is not only faster but has higher bandwidth, the size increase is paramount in workloads that take advantage of it, leaving the 3090 with no choice but to page it's memory to system RAM and take the performance penalty, which at times can be very severe.

C. Communication Hardware/Structure

The network fabric was deployed as follows:

- The Connect-X 6 200 Gb NIC are connected to the Infiniband fabric described in II-A1. One of the links is to the Ceph public network and will be used for storage access. The other link will be used to construct a cluster network for Proxmox, the details of which will be described in II-D.
- The Connect-X 4 Lx 25 Gb NIC are Ethernet uplinks to the rest of the datacenter infrastructure. They will carry all traffic to/from the Internet, as well as traffic destined for remote access to both the physical hardware and virtual machines that may be running.

D. Cloud Infrastructure Management

For our own cloud infrastructure management system/infrastructure provider, we deployed the hyper-converged[2] server management platform and enterprise virtualization solution Proxmox Virtual Environment (PVE) [10].
This platform was only deployed to compute nodes, so it will not be used in it's hyper-convergence capacity.
PVE has first-class support for Ceph, it being the storage system of choice for hyper-converged deployments. This allows PVE to provide Ceph (RADOS) block devices (RBD) as storage for virtual machines (VMs), as transparently as possible.
PVE is also a cluster system, so all physical machines are managed under a single control plane, so any virtual machines can be run transparently on any of the available physical machines, be transferred to any available physical machines (with restrictions) and be managed under a single interface. Unless specified otherwise, specifications of compute hardware from here on will always refer to virtualized setups, not bound to any single machine.

E. Work System

For executing our work, our first step will be defining the virtual machine template we will be using. This article will only reference the final specification, and since we are using VMs, the specifics of upgrading/deployment are not only hidden to us, but beyond the scope of this work.

- CPU: AMD EPYC 7443 24-core 2.85GHz (16 cores, virtualized)
- RAM: 128 GB
- 1x NVIDIA GeForce RTX 3090 (PCIe Passthrough[3])

Considerations about hardware specifications will be made as we consider the application we are running, since each has it's own requirements. For most of this work, we adopted a proof-of-concept approach to compute. For reasons to be explained later, we did not need to utilize multiple VMs for our workloads. However, we took the initiative to choose applications that could take advantage of multiple nodes, should we decide we needed them.

[2]Hyper-converged solutions are solutions that incorporate compute and storage on the same physical nodes, for cost savings and ease of deployment, configuration and maintenance.

[3]This is a virtualisation specific functionality allowing the use of the GPU as if it was part of a physical machine. The details of this functionality are beyond the scope of this work.

III. Data

A. Source - RIOT API

The Riot Games API [1], hereby known as Riot API, or simply the API, is an Application Programming Interface provided by Riot Games, Inc. for acquiring information about their games and matches playing on their platforms.
It's original purpose is to provide players with access to statistics and historical data about their playtime. In particular in relation to LoL, while the casual audience may not be interested in such details, those in the electronic sports (eSports) audience view this data as indispensable in their understanding of the game and their performance in tournaments.
We are interested in only some of the data available, as follows:

- Match data - This data includes aggregate information about each player and their performance in a match
- Rank data - The game features a competitive ladder mode, that places players within a given competitive division. The details of this system are out of scope for this work

The match data is organized by player, and can be agregated and analysed in multiple ways, to be described in sections IV, V. The rank data is most useful as a machine learning target. The details of its use are to be described in section V.

B. Storage - OpenSearch

For the applications we intend on using this data for, and considering the nature of the data itself, as well as it's format, it was clear that a traditional SQL database would not fit our requirements.
Our choice was therefore OpenSearch[4], a NoSQL document store database, originally designed for Enterprise Full-text search, and later augmented for analytics and observability workloads [9], as the underlying data store, Apache Lucene, was found particularly adept at the queries such workloads require.
It is also a distributed database system, scaling to thousands of nodes if required, with the ability to have different nodes execute different roles, for optimal resource utilization and performance.

1) System requirements: For our use, a properly sized OpenSearch cluster is a single node with 4 cores, 32 GB of RAM and storage as needed. Were we to have more data, or need to execute long-running or otherwise expensive queries, we would be able to multiply our cluster to multiple nodes distributed across the available infrastructure.

C. Preprocessing - Pipeline

In order to obtain and make available the data for use, several steps need to be completed. Assuming the infrastructure and database are ready, we proceed as follows:

- Obtain the data from the Riot API
- Check the data into the database
- Make the data available for use

[4]Open-source fork of Elasticsearch [5]

1) Data Acquisition: Data acquisition was performed in two stages, both using a dedicated ingestor program we built for the purpose, that can contact both the Riot API and the database.
The first version of this program (ingestor) operates as follows:

1) Start from a random player at the specified rank (Gold).
2) Add it to the queue.
3) Store that player information.
4) For all players x in the queue:
   a) Get their most recent ten matches.
   b) For each player y in the ten matches store them if they are not stored.
   c) If the player y is not in the database add him to the queue.
   d) Remove player x from the queue.

We allowed this program to run uninterrupted for $\sim$ 24 hours. In that time it collected 43,251 matches. From official sources [12], there are around 34 million players in the EU-West region. If we are to pull 10 matches from each, our current pace is much too slow. While it was decided that this amount was enough to continue working, this first ingestor program had a significant amount of improvements still to be made, and that we would be needing for the second iteration.
The second iteration of the program (ingestorv2) started with a different goal entirely. So far, we only collected match data, we still need rank data, and so the first job of ingestorv2 is to pull said data. It operates as follows:

1) Request a document iteration from the database. Iterate over all matches.
2) For every document returned by the database:
   a) Gather the players in said match.
   b) For every player in the match:
      i) If the database already has a rank for them, skip.
      ii) Otherwise, request a rank from the API.
      iii) Store the new rank in the database.

This new program took $\sim$ 48 hours to obtain 127,518 rank values, out of 130,186 unique player ids in the database. The discrepancy is justified as: a player may have played one or more ranked matches, but the system only assigns a rank after the start of a predetermined season, and the matches may be gathered from before the season start, and the player has not yet played any matches after the season rollover. Further analysis of this data was done in conjunction with other gathered data types in section V.
After completion of this data collection round, it became evident that, since the number of players is far higher than the number of matches, the number of matches per ranked player must be very low. Indeed, after a few calculations, it was determined that the average of matches per player equalled approximately 1.266, which is too low a number for an ideal Machine Learning application.
In order to improve the situation, we modified the ingestorv2 program and added the capability to add more matches to the database. It operates as follows:

1) Request a document iteration from the database. Iterate over all ranks.
2) For every document returned by the database:
    a) Count the number of matches said player appears in (this is a single, specialized database query)
    b) If the player is in 10 or more matches, skip.
    c) Request 10 additional matches from the API.
    d) For every returned match:
        i) As a sanity check, verify if the match was not already in the database). If it was, skip it.
        ii) Submit the match into the database.

One problem with this version of the program is that at this point we are fundamentally limited to the request rate we are allowed against the Riot API. To obtain our target of 10 matches per player, we would need to pull at most around 1.3 million matches, which we estimated to take over 21 days. Since we still need to complete other objectives in the project, the program was left to run undisturbed, and work continued on the data already collected.

2) Data indexing: Fruit of our selection of database engine, this section was made somewhat trivial. The data was inserted into the database with minimal to no processing required, as Opensearch accepts JSON documents for processing, the same format of data we receive from the Riot API. In total, three types of documents would be indexed:

- MatchDTO - Represents a single match
- SummonerDTO - Represents a single player
- LeagueEntryDTO - Represents a player's matchmaking rank in a given (ranked) queue.

Of the three types of documents obtained, we later found out SummonerDTO to be unnecessary, and would no longer attempt to collect that information. Regardless, no data was deleted and those documents already obtained were kept in the database.

As a side note, since we propose in section V to predict the rank of a single player, our collection of match and rank data was tailored to target only that specific subset of the total available data. Most importantly, only rank data pertaining to the 5x5 Solo Ranked Queue (queue id 420) was ever collected, as opposed to all ranks a player may have.

3) Data usage, visualization and analysis: While each of sections IV and V had to do their respective data processing and analysis, some of this work was shared among both sections, and for that reason is highlighted here. As part of Opensearch, our database of choice in this work comes the OpenSearch Dashboards, which not only serves as a frontend to the underlying OpenSearch database, it also provides data analysis and visualization tools, organized in "dashboards". To monitor and analyse the data we received in real-time (or not), we constructed a few dashboards of our own: Figure 8 shows a dashboard tracking the amount of rank data we currently have, as well as it's distribution and Figure 9 shows a dashboard tracking totals of match data, allowing us to monitor if our program is operating as we expect.

A few extra features include: Discover (Figure 10) for simply querying our data, and the Dev Tools (Figure 11) a direct, easy-to-use, HTTP query interface into Opensearch.

IV. Network Science

The work that this section refers to was built under the first 313673 players, originating from $\sim$ 50000 matches pulled from Riot. The network resulting from this construction had some undesired properties. It had a large number of cliques resulting from the mechanics of a match. That is, every player involved in a match connects to any other player in the match. This fact permits cliques of ten people to appear more than expected. In order to address this concern, we opted to condense the network by reducing the identified cliques within it. The initial stage entails the computation of all cliques in the network, employing the implementation provided by `networkit` [17]. We leveraged the efficiency of the algorithm and the multiprocessing approach to be able to efficiently compute said cliques.

After employing our implementation of a network condensation algorithm based on cliques we ended up with a network of 44587 nodes and 779615 edges. The network was nicknamed Network of Matches (NoM). The visualization of the network is presented in figure 1. Due to the size of the network, the only way it was possible to compute its representation was by leveraging the RAPIDS suite [18] of implementations of algorithms for GPU. In particular, we used the `cudf`, `cugraph` and `cuxfilter`. For the dynamic raster preview (the image in the report is static), we used Holoviews [16], Datashader and other tools to facilitate their integration. Analysis of large graphs using GPU, particularly `cugraph`, is very underdeveloped. Hence, most of the analysis was done using CPU alternatives.

We started by investigating the topology and dynamics of the network in order to discover what would be the mechanism that would be responsible for its generation. We tested against an initial null model to refute the random origin of the network. In order to efficiently do this, we relied on the multiprocessing implementations of both `networkit` and `graph-tool` [15]. These implementations are state-of-the-art software for large graph analysis. They are purely implemented in C++ and rely heavily on clever algorithms and multiprocessing to be efficient. After rejecting the null model, we studied multiple options that could originate the network. After fitting distributions and reasoning about the process, we concluded that the most likely process would be a Barabási-Albert process with initial attractiveness and node and link removal dynamics. Figure 2 shows the degree distribution of the NoM and the null models. Figure 3 shows the difference in the normalized centralities for an Erdos-Renyi random model and NoM.

After discussing the topology of the network we conducted tests regarding inverse bound and site percolation to see the resilience of the network and tests regarding diffusion models to test the propagation of information.

With percolation, we wanted to test if it was possible to tune the initial attractiveness parameter that modifies $k_{\min}$ in order to improve the resilience of the network. Figure 4 and

5 show the effect of the different $k_{\min}$s. We concluded that a higher $k_{\min}$ achieves increased robustness.

We also studied different processes of information diffusion in order to understand how attempts of scams may propagate through the network. Figure 6 and and 7 show the progress of the scam epidemic. More details can be found in C. Vieira and Afecto [6].

V. Machine Learning

In this segment, we will focus on machine learning engineering and compare the performance of CPU and GPU-bound ML algorithms using the large amount of data collected in previous sections. Additionally, we will explore the OpenSearch Python API [9] for data aggregation purposes.

To begin, we will utilize `plotly` Dashboards [11] in conjunction with OpenSearch queries in Python to visualize and explore the data. Next, we will aggregate the data, allowing us to compile a comprehensive dataset. This dataset will be used for further analysis, such as Exploratory Data Analysis (EDA) and benchmarking of inference models.

For such models, the defined target variable consists on the tier of each player registered in the data source given their matches information.

A. Data Visualisation

An important part of the study conducted on Machine Learning models is the API connection established between the provided data source and the main engine of the models, for this specific case: the OpenSearch engine and the Python libraries, made through OpenSearch Python Client. To get started with such technology and get a brief introduction to the data, a DashBoard (apart from the one referred to on III-C3) was developed in Python using `plotly` and a Dashboard module.

The Dash module allows for dynamic changes in the displayed plots based on an interactive User Interface (UI) on HTML, for this reason, a dynamic query builder was developed and used alongside OpenSearch's Python Client, granting the possibility of real-time updates to the plots through directly communicating with the data source on the cloud.

The plots consist mainly of a comparison between the performance of the two teams presented in the game, displaying information such as the average number of wins and objectives contested as displayed in figure 12. The dashboard also has the average match time duration and total resource values gathered through the game (gold and experience) as can be seen in figure 13. Furthermore, it collects character picks and "first blood" (the name given to the first kill of the game) as observed in figure 14. Such plots would support the idea of a balanced and unbiased data set.

For exploration purposes, the dashboard was also equipped with said tools for dynamic plot updates. With said tools, one could filter teams, regions, queues and game types as well as the size and dates of matches as displayed in figure 15. With the dynamic query system built, the refereed plots are all online and coherent with the data from the source. The number of observations is also dynamic as it follows directly the result of *runtime* queries.

The same dashboard page developed will proceed to be also the "home" of the EDA and benchmarks to be presented in subsections V-C and V-D.

B. Data Aggregation and Replicability

The main objective as once expressed (at V) will be to compare multiple models of Machine Learning bound by both CPU and GPU, this way some benchmarks of time and metrics will be displayed. Given the execution of tests and given their reproducibility, some level of data stability may be reached considering the fluctuations of the data source in both volume and diversity.

This way, before executing the tests, the data source was queried for the current data, which was then processed and stored in a CSV document, ensuring the data used on each test was perfectly equal for every model and platform.

The pre-processing of such data consisted of an aggregation made yet at the data source level, considering the existence of multiple matches associated with each player, and the variety of performances exhibited by each of them, the matches were grouped by the presence of a player, and queried by both player stats and match stats gathering different aspects like average damage dealt, gold earned and win ratio.

The compressed data set was split into Test and Train set at this level and saved on different CSV files, ensuring a higher reproducibility between GPU and CPU platforms, considering the different packages.

C. EDA and Semantics

Before one hops into the statistical inference of the data, one must understand some relations and variables presented in such. While League of Legends is a complex game with several hidden and deep relations described in it, this segment will consider some Exploratory Data Analysis and try to display some of the semantics of the data.

For an initial understanding, a brief introduction to League of Legends Mechanics is presented, leading the reader to a basic understanding of some of the core principles of the analysed data.

League of Legends is a PC *Multiplayer Online Battle Arena* (MOBA for short), the main idea of the game consists of two teams, each consisting of five players, who compete against each other intending to destroy the enemy team's Nexus, a structure located at the heart of their *base*. A team base is a region of the map where players and AI-controlled minions can *spawn*, they are located on the corners of the map and accessible through 3 possible pathways, called the three main *lanes*: top, middle, and bottom. Each lane has protective structures, that only attack enemy players and minions, called *Turrets*. Turrets can be destroyed, enabling way of players and minions to traverse the map and attack the enemy's nexus.

Each player assumes control of *champion*, a unique character with distinct abilities and play styles, their primary objective is to guide minions, gain map control by destroying

turrets, eliminating enemy champions, and securing objectives till the enemy nexus is exposed enough to be attacked. To achieve this objective, players engage in combat with both enemy champions and regularly spawned waves of AI-controlled minions, successfully defeating minions and other players, rewards players with *gold* and *experience*, enabling the progression of their champions.

Gold, a vital resource in the game, as mentioned it can be earned through various actions, including slaying minions and enemy champions. It allows players to purchase items from the in-game shop, enhancing their champion's abilities and attributes, such as damage, defence, and utility, providing a significant advantage in battles, and allowing players to outtrade and outperform their opponents.

Experience is another main factor of the game, players accumulate experience killing AI-controlled minions, and with certain thresholds the players' champion level up, unlocking new abilities and improving existing ones. As players gain levels, their champions become more powerful, gaining increased base statistics and access to stronger spells, such progression grants players additional tools to dominate fights and adapt to different situations.

Besides the three main pathways, a fourth location is crucial in the game: the jungle. It serves as a transitional space between the three lanes, where players can acquire additional resources and exert map control, containing various *camps*, locations where AI-Controlled monsters may appear, those monsters are neutral and only attack players when attacked by them, such monsters also provide gold and experience similar to the minions. However, some jungle camps, called epic monsters, provide additional resources granting a player more gold, experience and special *buffs*, like the Blue and Red Buff. Other significant monsters in the jungle include the Dragon and the Herald, those monsters appear in specific locations on the map and provide even higher buffs and advantages for the team that slain them.

For the provided data set and subsequent aggregation, 44 numerical variables were chosen for a total of 127518 player entries, most of the variables consisting in average values per match of selected stats, such as damage dealt to champions, objectives or buildings, it was also considered important the duration of the match and the play style selected by the player and communication pings, values that affect the metadata of the game and may display some correlation with both the rank and the variables chosen. For example, players with mostly "Support" play style may exhibit more communication, healing and shielding behaviour, with less aggressive stats like damage dealt. In the same way, a player with a more "Jungle" play style may hold greater damage to objectives and Jungle Minions.

This high diversity of players and matches, combined with the high complexity of the variables' interactions may prove that predicting a player's rank by their matches stats will not be an easy task. A fact blackened up by the plot 16 where it becomes clear that the distributions of the variables for each rank are almost perfectly overlapped.

For the correlations plot, displayed at figure 17, some interesting patterns can be spotted in the middle of the plot: a "square" shape of higher correlations can be seen, those are mainly related to the total time of the match, compressing total damage, healing, kills and their respective consequences over time. Denser squares can be spotted, related to kills, damage and total gold collected (relating to the clear relation between the game economy and player performance) as well as objective damages and epic monsters killed, mainly related to the jungle player presence. interestingly, one of the variables is highlighted in the middle of the biggest square, "*totalDamageShieldedOnTeammatesAvg*" presents lower correlations on such square.

Finally, histograms plot (figure 18) are presented, some distributions have close to a symmetric shape and some have a high skew presence. Many of the variables have an implicit distribution usually at the left, indicating short games usually ended on a team surrender at 15 minutes. The variables should be scaled.

D. Model Benchmarks

This section will be dedicated to exploring Machine Learning models developed with both the Scikit Learn package [14] from Python as the CPU-bound interface and RAPIDS *cuml*, as the GPU one, the XGBoost Python package [8] was also used, having both CPU and GPU interface. Take into account that all the CPU methodologies were parameterised to use all available threads. The machine used for both GPU and CPU tests was the one previously referred at section II-B.

For additional comparisons, a secondary machine was used, representing a day-to-day computer outside the cloud, it will be further referred to in this work as the "*local*" machine, it was equipped with a stock AMD Ryzen 5800X @ 3.8 GHz up to 4.7 GHz and 2x DDR4 3200MHz 8GB.

The comparison for every model will be made on two different sides, first and most important is the time performance on both training and predicting, next the classification performance of the algorithms running on different platforms will be explored, in principle the algorithms are mostly deterministic (or made deterministic by setting a random seed) so no great variations should be expected between models. This comparison was mostly made in order to testify to the integrity of the tests, an important step later on discussed.

1) Model choices and notes: The main motivation for the choices of models was the availability of an equivalent model on both platforms (CPU and GPU), the algorithms chosen were present in both SciKit Learn and RAPIDS or yet had their proper package available for both GPU and CPU. The list of packages is described below:

- XGBoost (own package);
- Support Vector Machines (SciKit and RAPIDS);
- Random Forest (SciKit and RAPIDS);
- K-Nearest Neighbours (SciKit and RAPIDS);
- Logistic Regression (SciKit and RAPIDS);

Some aspects are important to be appointed after further analysis of the presented methodologies:

- The Support Vector Machines (SVM) implementation for SciKit Learn is single-threaded, producing very slow results;
- After close inspection of classification metrics, it was noticeable that the "class_balance" (from our earlier analysis, most likely a bug on Rapids) parameter for the Logistic Regression did not affect the model performance as expected and observed on the SciKit version of the method, for this reason, the unbalance version was adopted for the Logistic Regression, to standardise the results;
- The XGBoost model, due to its good performance on a preliminary test was tested with 2 different values (6 and 10) for its maximum depth;

2) Results: As expected the result of this experiment demonstrated the power of the GPU platform when compared to the CPU bound of the same methodology, as presented in the plot 19 it is clear that the GPU has an advantage in both training and predicting on every model that involves mathematically heavy operations and searches. The advantage was only not visible on KNN training, a task that does not require operations at all but storage, the advantage became visible on the same method by the prediction time, as expected with an almost 20x faster prediction. The SVM results got to be isolated due to its scale, those results were also expected given the Single Threaded action of the SciKit package. It is worth noting the bad capacity of the GPUs for predicting Random Forest values, this may be due to a more intense synchronous task rather than a parallel search. The local machine, as expected, did not outperform the other platforms in any task, with a single exception of the SVM in training, this outlier may be due to the single thread aspect of the task, in which the local CPU outperformed the cloud due to its higher clock speed (3.8 GHz vs 2.85 GHz).

The prediction metrics are displayed on plots 20 to 27, and the results are outstanding, this task was hard (statistically speaking) and the unbalance in the data set made it worse. An important appointment from these plots is 26 and 27, both refer to the Logistic Regression, one with class balancing and the other without it, there becomes visible the lack of impact from the weight in the Logistic Regression at RAPIDS as referred in the segment V-D1. From all the models, XGBoost with a maximum depth of 6 presented the best results and a good balance of training and predicting a time when executed on a GPU, where the time gain was very significant (almost 15x faster).

VI. Conclusion

All the code, reports and other details can be found in https://github.com/luxcorppt/LoLMLACN.

Acknowledgements

We would like to express our deepest appreciation to Paulo Ramos, Rolando Martins, Inês Dutra and Eduardo Marques for giving us the opportunity, the knowledge and the required permissions to mount and deploy this infrastructure from scratch.

Many thanks to Rolando Martins once again, for letting us mount, configure and deploy his hardware.

Special thanks to Pedro Ribeiro, Inês Dutra and Eduardo Marques once again, for allowing us to join assignments from different courses into this one single work.

We'd like to recognise Tiago Eusébio for helping us better understand the game League of Legends as our subject matter expert.

We would be remiss in not mentioning the Department of Computer Science, Faculty of Science of University of Porto for allowing us the privilege of working along side System Administrators and Professors.

APPENDIX

A. Network Science

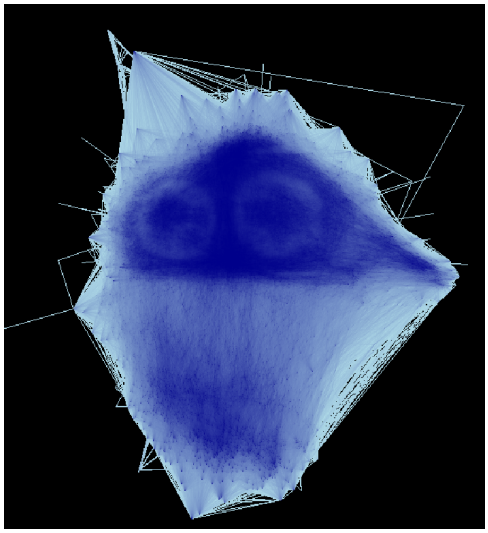

Fig. 1. Visual representation of NoM . Dark blue points represent nodes and light blue edges.

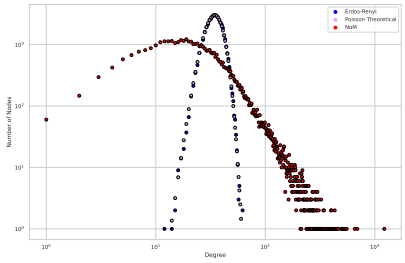

Fig. 2. Degree distribution plotted on log-log scale for NoM, a network generated by the ER model with parameters to match NoM and the theoretical approximation of the distribution of such networks.

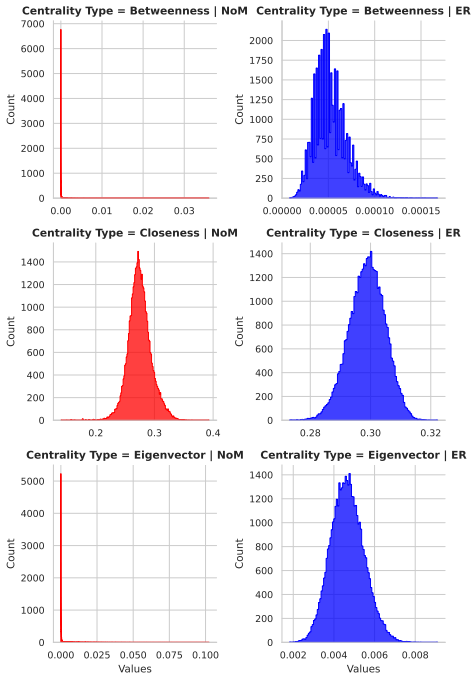

Fig. 3. Normalized centralities for NoM and ER network with parameters matching NoM.

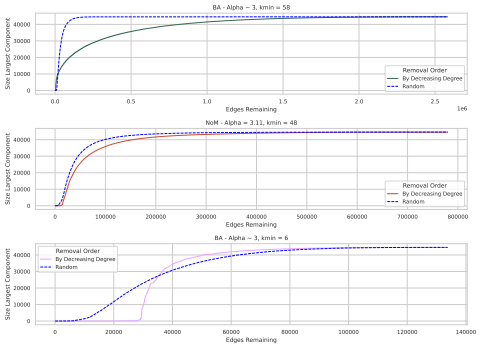

Fig. 4. Evolution of the size of the largest component as nodes are removed of the network. A null model consisted of random removal and a attack model consisting of removing nodes starting from the ones with a higher degree are compared.

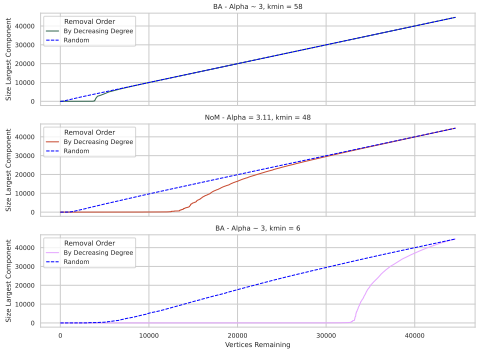

Fig. 5. Evolution of the size of the largest component as edges are removed of the network. A null model consisted of random removal and a attack model consisting of removing edges starting from the ones with a higher weight are compared. Weight was defined by the multiplication of the degree of the nodes that an edges connects.

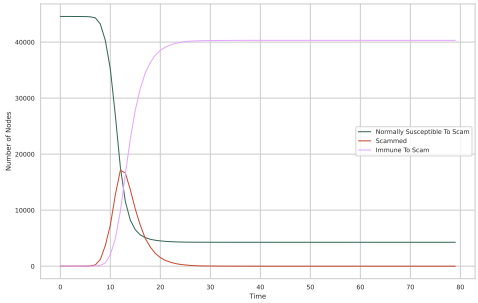

Fig. 6. Evolution of the state of the nodes of NoM according to the diffusion model SIR.

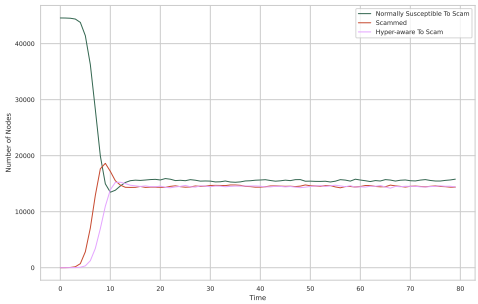

Fig. 7. Evolution of the state of the nodes of NoM according to the diffusion model SIRS.

B. Data

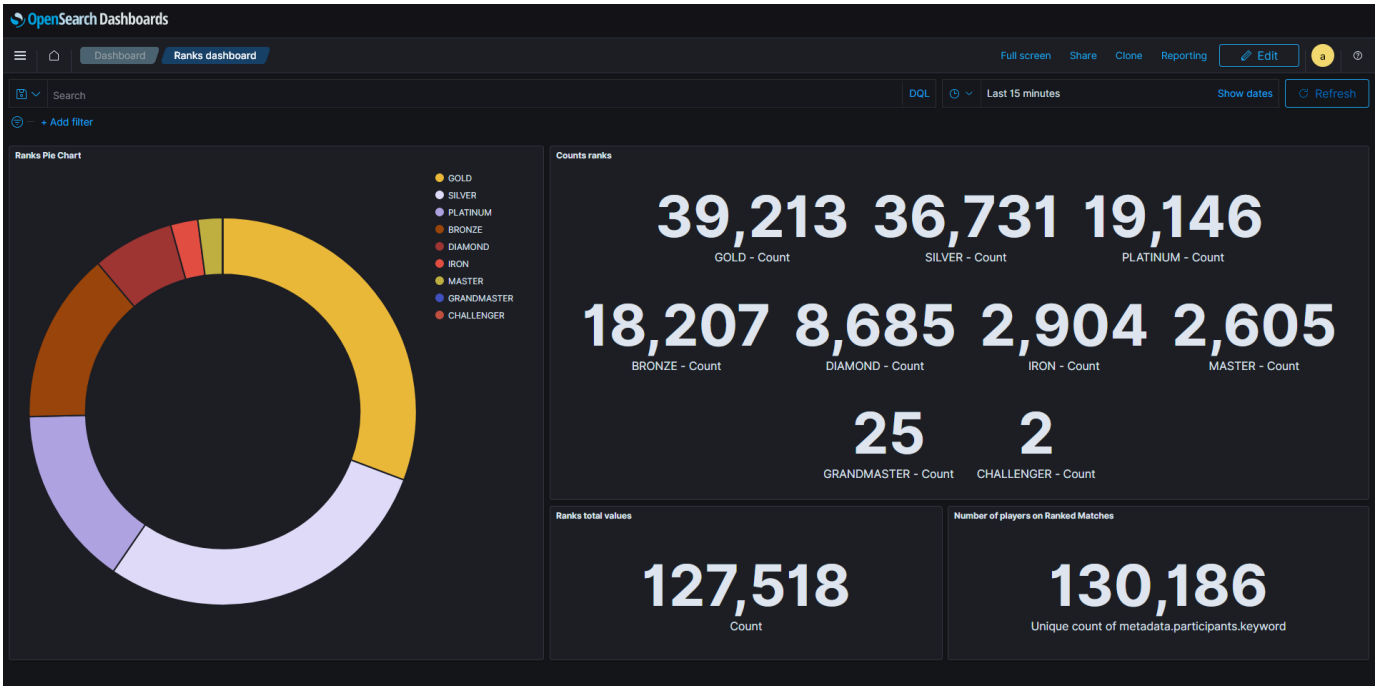

Fig. 8.

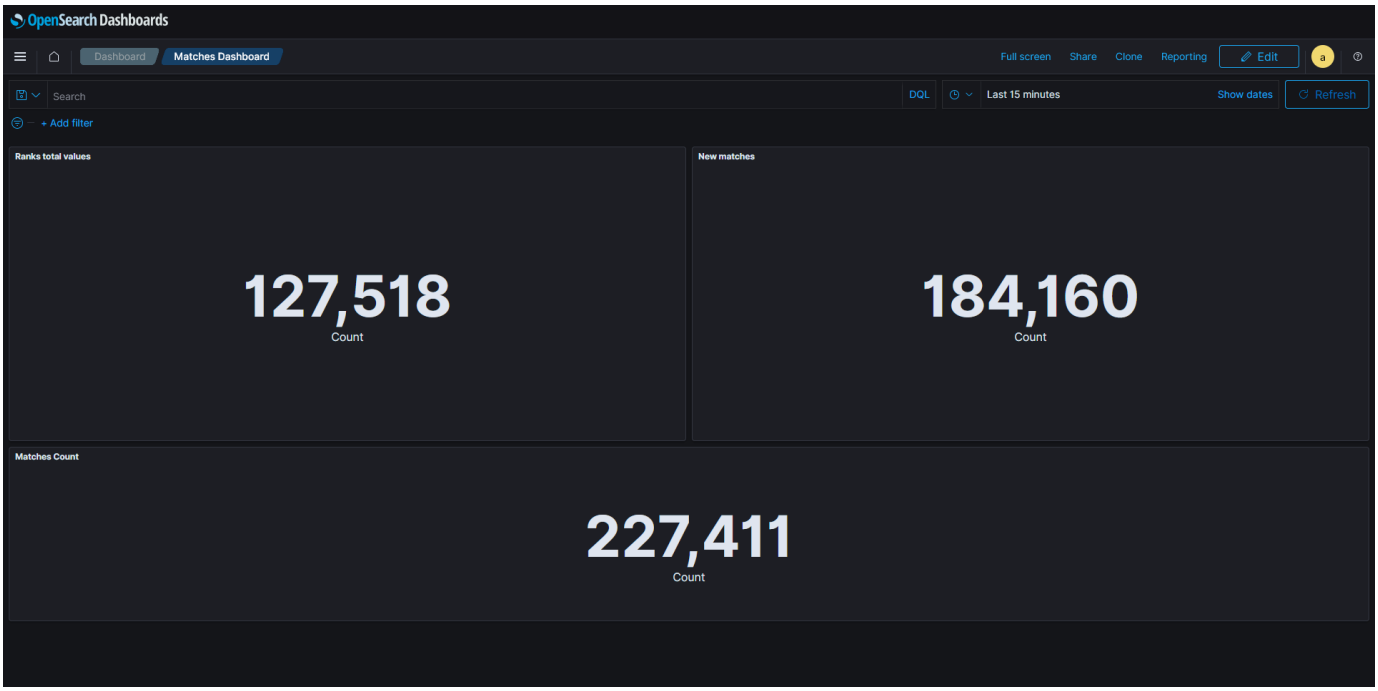

Fig. 9.

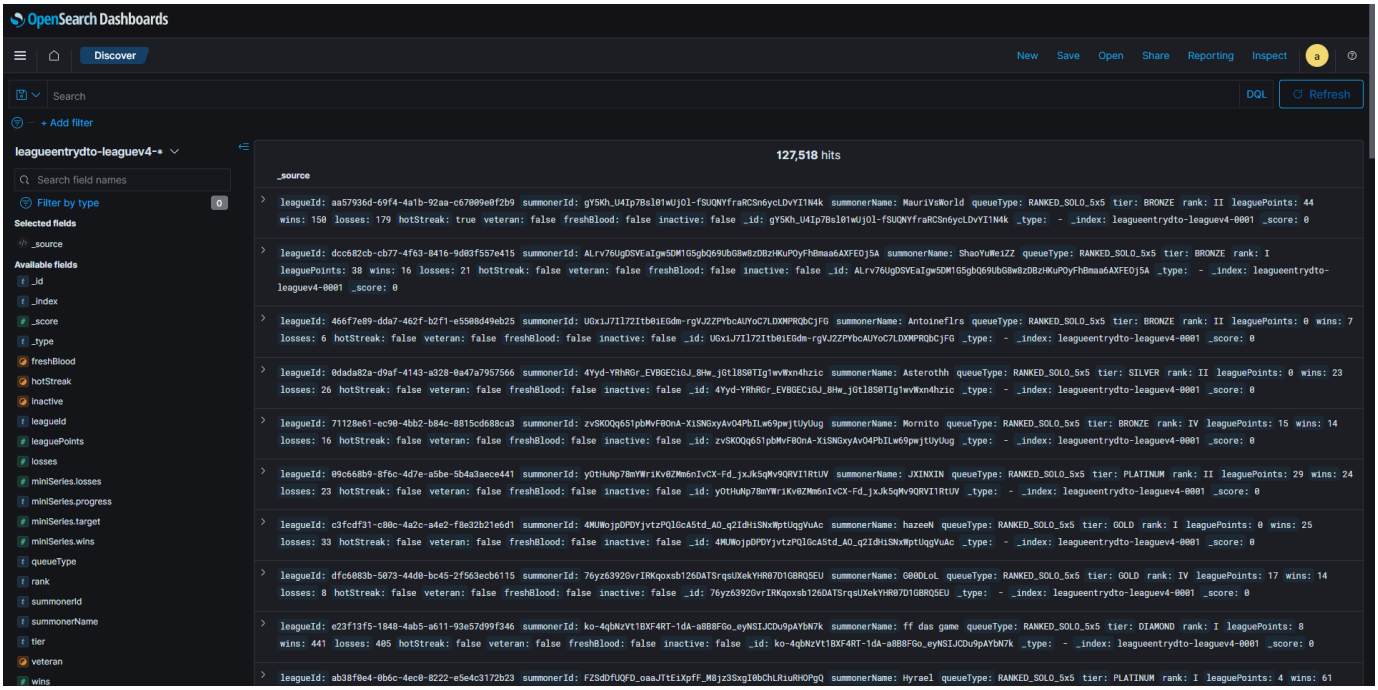
OpenSearch Dashboards
Discover
New
Save
Open
Share
Reporting
Inspect
Search
DQL
Refresh
+ Add filter
leagueentrydto-leaguev4-*
Search field names
Filter by type
Selected fields
_source
Available fields
_id
_index
_score
_type
freshBlood
hotStreak
inactive
leagueId
leaguePoints
losses
miniSeries.losses
miniSeries.progress
miniSeries.target
miniSeries.wins
queueType
rank
summonerId
summonerName
tier
veteran
wins
127,518 hits
_source

Fig. 10.

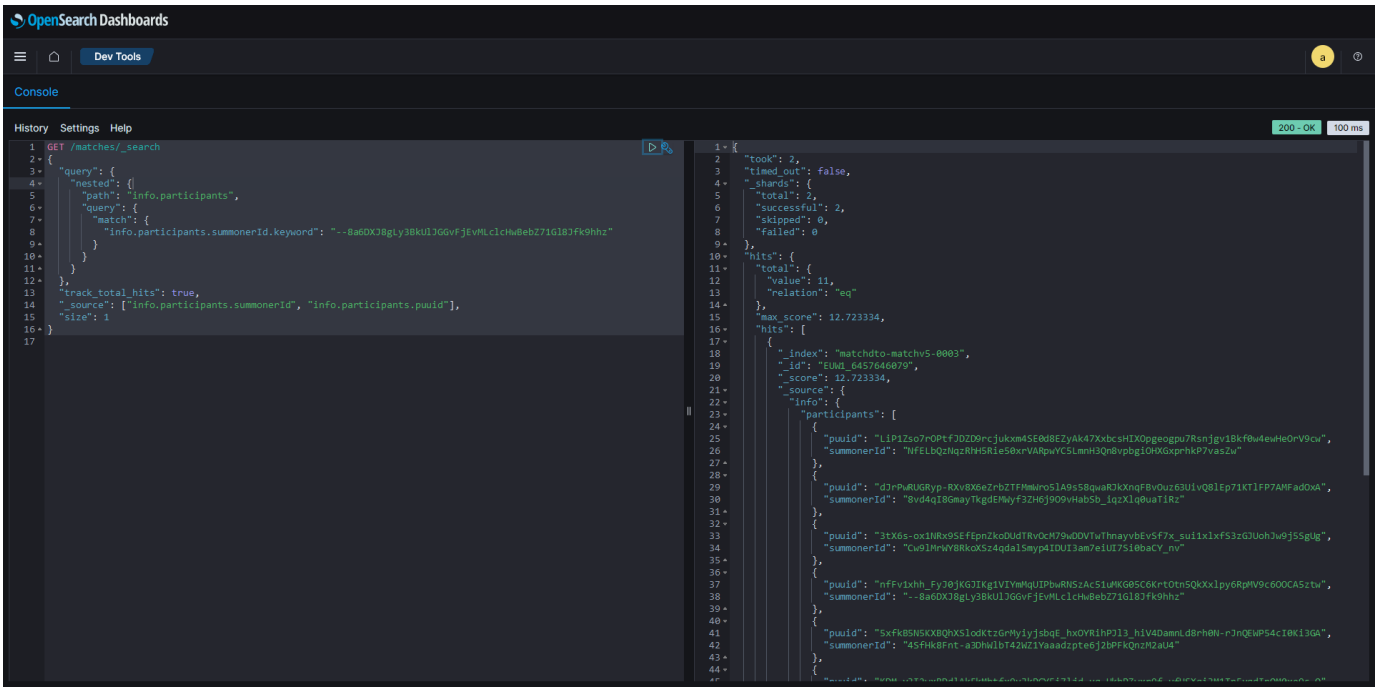
OpenSearch Dashboards
Dev Tools
Console
History
Settings
Help
200 - OK
100 ms
GET /matches/_search
{
"query": {
"nested": {
"path": "info.participants",
"query": {
"match": {
"info.participants.summonerId.keyword": "--8a6DXJ8gLy3BkUlJGGvFjEvMLclcHwBebZ71Gl8JFk9hhz"
}
}
}
},
"track_total_hits": true,
"_source": ["info.participants.summonerId", "info.participants.puuid"],
"size": 1
}
{
"took": 2,
"timed_out": false,
"_shards": {
"total": 2,
"successful": 2,
"skipped": 0,
"failed": 0
},
"hits": {
"total": {
"value": 11,
"relation": "eq"
},
"max_score": 12.723334,
"hits": [
{
"_index": "matchdto-matchv5-0003",
"_id": "EUW1_6457646079",
"_score": 12.723334,
"_source": {
"info": {
"participants": [
{
"puuid": "LiP1Zso7rOPtfJDZD9rcjukxm4SE0d8EZyAk47XxbcsHIXOpgeogpu7Rsnjgv1Bkf0w4ewHeOrV9cw",
"summonerId": "NfELbQzNqzRH5Rie50xrVARpwYC5LmnH3Qn8vpbgiOHXGxprhkP7vasZw"
},
{
"puuid": "dJrPwRUGRyp-RXvBX6eZrbZTFMmWro5lA9s58qwaRJkXnqFBvOuz63UivQ8lEp71KTlFP7AMFadOxA",
"summonerId": "Bvd4qI8GmayTkgdEMwyf3ZH6j9O9vHab5b_lqzXlq0uaTiRz"
},
{
"puuid": "3tX6s-ox1NRx9SEfEpnZkoDUdTRvOcM79wDDVfwThnayvbEvSf7x_sui1xlxfS3zGJUohJw9jSSgUg",
"summonerId": "Cw9lMrWY8RkoX5z4qdalSmyp4IDUI3am7eiUI7Si0baCY_nv"
},
{
"puuid": "nfFv1xhh_Fy30jKGJIKg1VIYmMqUIPbwRNSzAc51uMKG05C6KrtOtn5QkXxlpy6RpMV9c6OOCA5ztw",
"summonerId": "--8a6DXJ8gLy3BkUlJGGvFjEvMLclcHwBebZ71Gl8JFk9hhz"
},
{
"puuid": "5xfkB5N5KXBQhXSlodKtzGrMyiyjsbqE_hxOYRihPJl3_hiV4DamnLd8rh0N-rJnQEWP54cI0Ki3GA",
"summonerId": "45fHk8Fnt-a3DhWlbT42WZ1Yaaadzpte6j2bPFkQnzM2aU4"
},

Fig. 11.

C. Machine Learning

Fig. 12. Plots comparing wins and objectives from each of the teams in a Match, dynamically updated on the dashboard

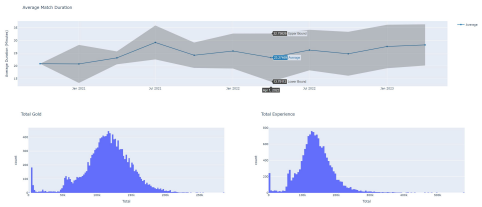

Fig. 13. Plots of average duration of Matches grouped by period of time, total gold and experience dynamically updated on the dashboard

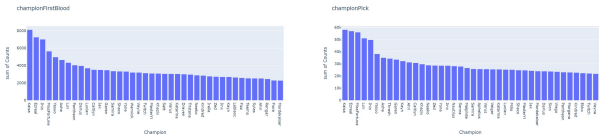

Fig. 14. Plots of Champion Pick rate and First Blood dynamically updated on the dashboard

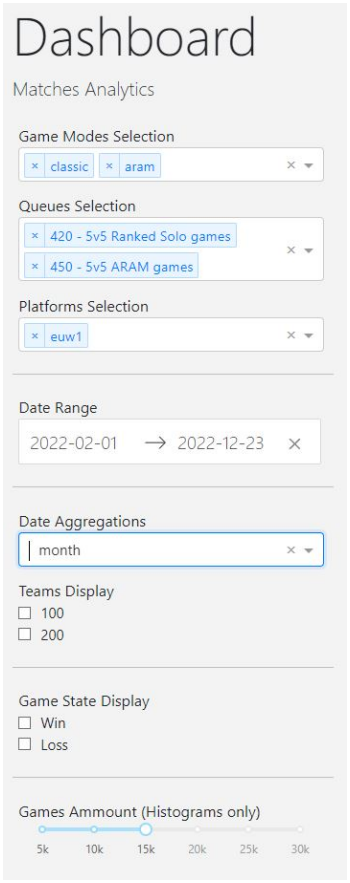

Fig. 15. Dashboard control panel with the options for dynamic changes on the query to the data source

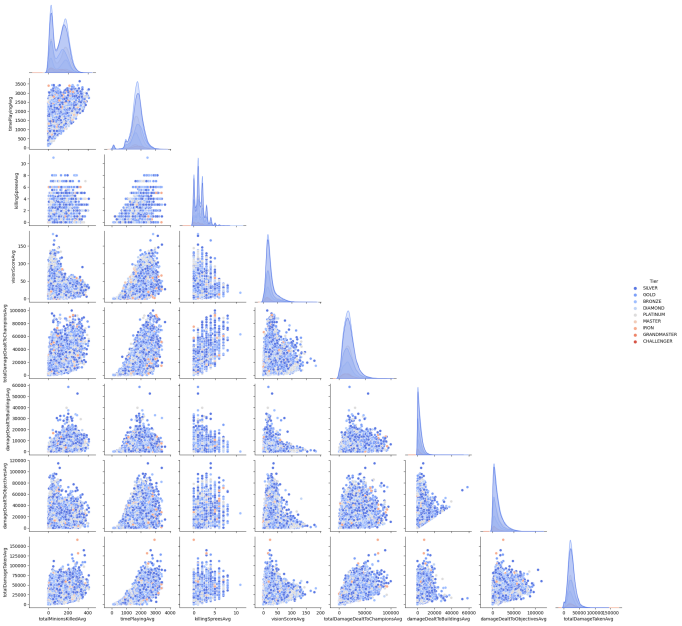

Fig. 16. Pairplot with some variables and a 5000 sample from the aggregated data

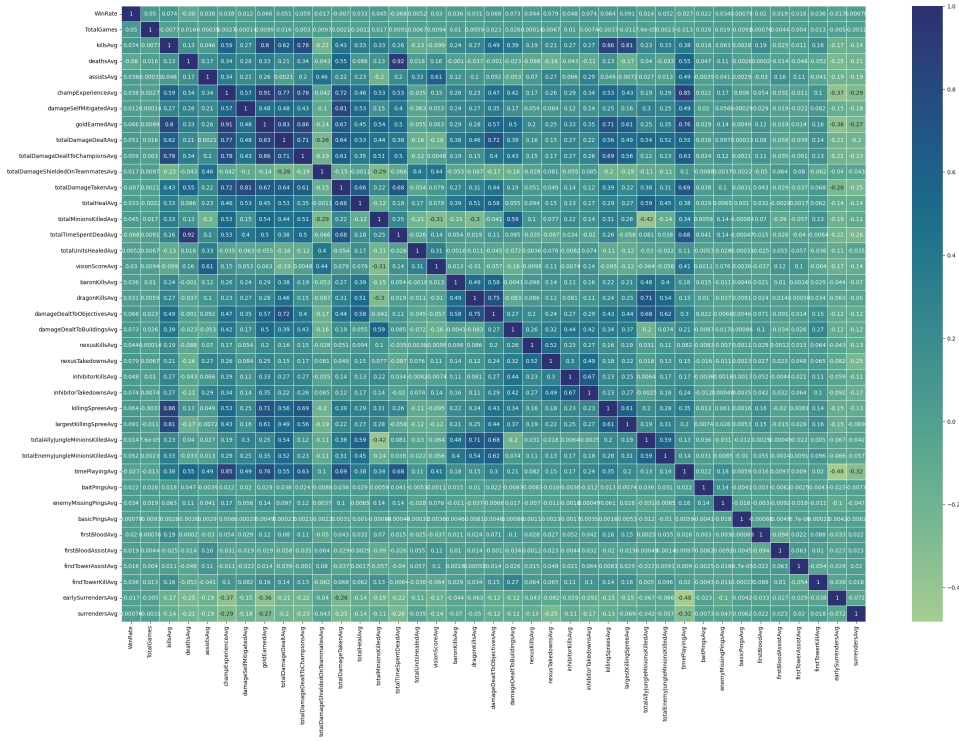

Fig. 17. Correlation plot from the aggregated data

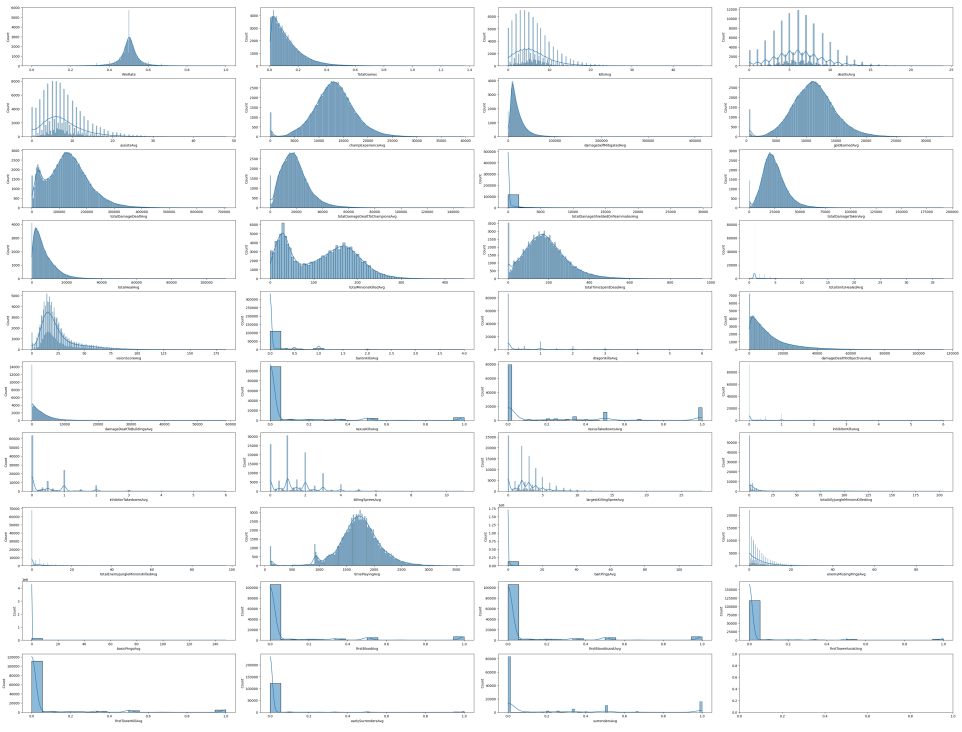

Fig. 18. Distributions of the variables from the aggregated data

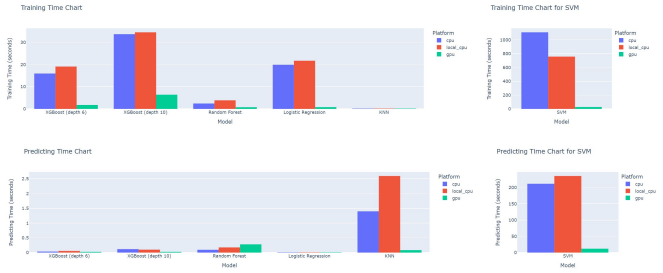

Fig. 19. Time comparison in Train and Prediction for each Machine Learning Model at all platforms

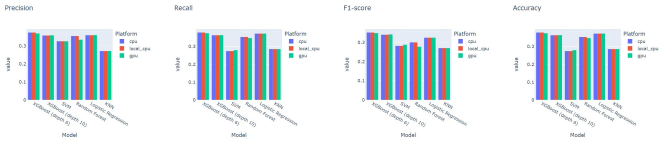

Fig. 20. Average metrics comparison for every Machine Learnin Model, ploted against each other in each platform

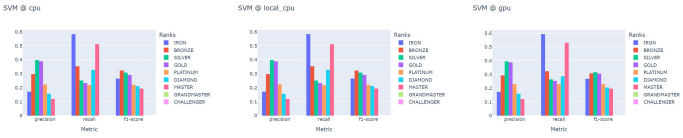

Fig. 21. Metrics Comparison grouped by rank on every platform for SVM

Fig. 22. Metrics Comparison grouped by rank on every platform for XGBoost with depth 6

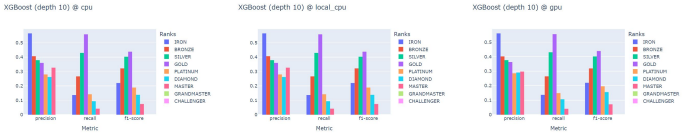

Fig. 23. Metrics Comparison grouped by rank on every platform for XGBoost with depth 10

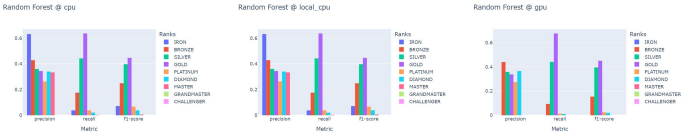

Fig. 24. Metrics Comparison grouped by rank on every platform for Random Forest

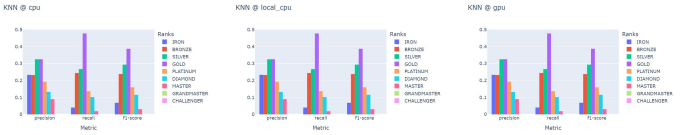

Fig. 25. Metrics Comparison grouped by rank on every platform for KNN

Fig. 26. Metrics Comparison grouped by rank on every platform for Logistic Regression without class weights

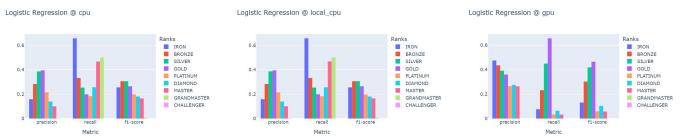

Fig. 27. Metrics Comparison grouped by rank on every platform for Logistic Regression with class weights

REFERENCES

[1] Developer page riot games. Accessed on 2023-06-13, https://developer.riotgames.com.

[2] Support forum league of legends. Accessed on 2023-06-13, https://support-leagueoflegends.riotgames.com/hc/en-us/articles/201751684-League-of-Legends-Regional-Servers.

[3] Ceph authors and contributors. Ceph documentation. Accessed on 2023-06-26, https://docs.ceph.com/en/quincy/.

[4] Ceph authors and contributors. Ceph.io - discover. Accessed on 2023-06-26, https://ceph.io/en/discover/.

[5] Elasticsearch B.V. What is elasticsearch? — elastic. Accessed on 2023-06-26, https://www.elastic.co/what-is/elasticsearch.

[6] Pedro C. Vieira and João Afecto. Dynamic Mechanisms forming Networks of MOBA Matches : Understanding Overlaps , Topology , Resilience and Diffusion in a Gaming Community. Technical report, Department of Computer Science, Faculty of Sciences University of Porto, 2023.

[7] Alejandro Cannizzo and Esmitt Ramírez. Towards procedural map and character generation for the MOBA game genre. *Ingeniería y Ciencia*, 11(22):95–119, July 2015.

[8] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[9] OpenSearch contributors. About opensearch. Accessed on 2023-06-26, https://opensearch.org/about.html.

[10] Proxmox Server Solutions GmbH. Proxmox ve - virtualization management platform. Accessed on 2023-06-26, https://www.proxmox.com/en/proxmox-ve.

[11] Plotly Technologies Inc. Collaborative data science, 2015.

[12] Rijad Kamberovic. League of legends player count: Here are the stats. Accessed on 2023-06-13, https://riftfeed.gg/more/player-count.

[13] NVIDIA & Mellanox. Switch system product brief - qm8790 mellanox quantum™hdr edge switch. Accessed on 2023-06-26, https://network.nvidia.com/files/doc-2020/pb-qm8790.pdf.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[15] Tiago P. Peixoto. The graph-tool python library. *figshare*, 2014.

[16] Philipp Rudiger, Julia Signell, James A. Bednar, , Andrew, Jean-Luc Stevens, Chris B, Jordan Samuels, , Todd, Thomas PEDOT, Sander Van Den Oord, Jon Mease, Isaac Virshup, Gabriel Corona, Danny Hermes, H. Curtis, and Anita Graser. holoviz/hvplot: Version 0.5.2, 2020.

[17] Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. Networkit: A tool suite for large-scale complex network analysis, 2014.

[18] RAPIDS Development Team. *RAPIDS: Libraries for End to End GPU Data Science*, 2023. https://rapids.ai.

[19] TechPowerUp. Nvidia geforce rtx 3090 specs — techpowerup gpu database. Accessed on 2023-06-26, https://www.techpowerup.com/gpu-specs/geforce-rtx-3090.c3622.

[20] TechPowerUp. Nvidia rtx a6000 specs — techpowerup gpu database. Accessed on 2023-06-26, https://www.techpowerup.com/gpu-specs/rtx-a6000.c3686.

[21] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, and Carlos Maltzahn. Crush: Controlled, scalable, decentralized placement of replicated data. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, page 122–es, New York, NY, USA, 2006. Association for Computing Machinery.

[22] Sage A. Weil, Andrew W. Leung, Scott A. Brandt, and Carlos Maltzahn. Rados: A scalable, reliable storage service for petabyte-scale storage clusters. In *Proceedings of the 2nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing '07*, PDSW '07, page 35–44, New York, NY, USA, 2007. Association for Computing Machinery.