VCT Esports Manager Challenge

Kyle Zhou, Yufan Yang

October 24, 2024

Abstract

This project explores the application of Large Language Models (LLMs) combined with Gaussian Mixture Modeling (GMM) and Principal Component Analysis (PCA) to generate optimal team compositions for the Valorant Championship Tour (VCT). The approach leverages LLMs for analyzing player data, performance metrics, and strategic synergies, while GMM and PCA is employed to reduce dimensionality and identify key factors that influence player compatibility and team dynamics. This integrated method offers recommendations tailored to competitive play, balancing individual skills with team cohesion. Results indicate that the LLM and PCA-based system enhances decision-making, offering data-driven insights that streamline the process of forming high-performance VCT teams. This project highlights the potential for AI-assisted tools in the esports industry, aiming to improve team selection efficiency and performance through advanced data analysis.

1 Introduction

VCT represents the pinnacle of competitions, where the best of the best compete against each other and show off their skills. Current methods of team formation rely heavily on the experience and intuition of managers, who assess players' skills, roles, and synergy through a time-consuming process. While statistical analysis has long been a part of traditional sports, the esports scene seems to be a bit behind in this aspect, likely due to the large cost overhead it requires to maintain scouting teams. The objective here is to find a way to combine well studied classical methods with the natural language capabilities of an LLM to make stats driven team building more accessible and cost-effective to the Valorant community.

2 Methodology

Considering that no one on our team has ever worked with AWS before, has experience building LLMs/transformer applications, and that we are a very late entry into the competition, we have one main constraint that we need to design around: we need to build this product in 2 weeks. We settled on an implementation that would blend well understood unsupervised learning methods with the cutting edge LLM toolbox in AWS Bedrock. This serves to lower the amount of time spent tagging data and wrestling with new, complex concepts as well as to buy some time to learn the AWS tango and create a quality final product. That being said, there are a couple core assumptions that we will operate under:

1. Valorant teams are always comprised of 5 roles: Duelist, Sentinel, Initiator, Controller, and a Flex player.
2. Role patterns found in VCT International Data will be consistent and applicable to other skill brackets (ie VCT Challengers and Game Changers).
3. Players that make it to VCT Champions are the best players in their respective roles.
4. There exists some n-dimensional vector space V of all valorant players.
5. Any given player's vector $\vec{p}$ can be approximated by aggregating that player's statistics on a per round basis.

Ultimately this project can be divided into two stages: data mining and application creation. The idea is to use GMM and PCA to isolate the best players at each role and from each league and region, generate probability distributions on a per role, per league, per region basis, and sample out of these distributions to generate teams. At this point we can pass the teams to an LLM for context aware filtering and analysis until we end with the "best" team. Then it is a matter of presenting an LLM session to the user with a natural language breakdown of this team consistent with the challenge requirements:

- Answer questions about player performance with specific agents (in-game playable characters)
- Assign roles to players on the team and explain their contribution
  - Offensive vs. defensive roles
  - Category of in-game playable character / agent (duelist, sentinel, controller, initiator)
  - Assign a team IGL (team leader, primary strategist and shotcaller)
- Provide insights on team strategy and hypothesize team strengths and weaknesses

2.1 Data mining

2.1.1 Gaussian Mixture Modeling

We start our data mining by looking at the VCT International Dataset. This is because it is the sampling of the best players in Valorant as well as the one we have the most familiarity with in order to sanity check results. As mentioned before our data mining follows a very typical classification workflow, with clustering being a natural place to start. Since we have no ideas on what our data's shape is, we decided to do probabilistic clustering rather than distanced based clustering. Interestingly in our tests, we found Bayesian GMM (BGMM) to generate more meaningful clusters than traditional GMM even when limited to the same number of clustering groups. So we decided to use a BGMM model.

Before we do clustering we first need to define the components of $\vec{p}$ to best capture a player's profile. Here its important to remove any personal intuition about what makes a good player good so as to minimize personal bias fitting during the unsupervised learning process. While this intuition is valuable, it can be mixed in to the agnostic output of our models later and with more precision. To this end we have selected the following raw statistics to represent $\vec{p}$:

- ASSISTS - Average number of assists per round
- ASTRA_PICK_RATE - Percent rounds played as Astra
- BREACH_PICK_RATE - Percent rounds played as Breach
- BRIMSTONE_PICK_RATE - Percent rounds played as Brimstone
- CHAMBER_PICK_RATE - Percent rounds played as Chamber
- CLOVE_PICK_RATE - Percent rounds played as Clove
- CYPHER_PICK_RATE - Percent rounds played as Cypher
- DAMAGE_DONE - Average damage done per round
- DAMAGE_TAKEN - Average damage taken per round
- DEAD - Likelihood of dying in a round
- DEADLOCK_PICK_RATE - Percent rounds played as Deadlock
- DEATHS - Average deaths per round
- FADE_PICK_RATE - Percent rounds played as Fade

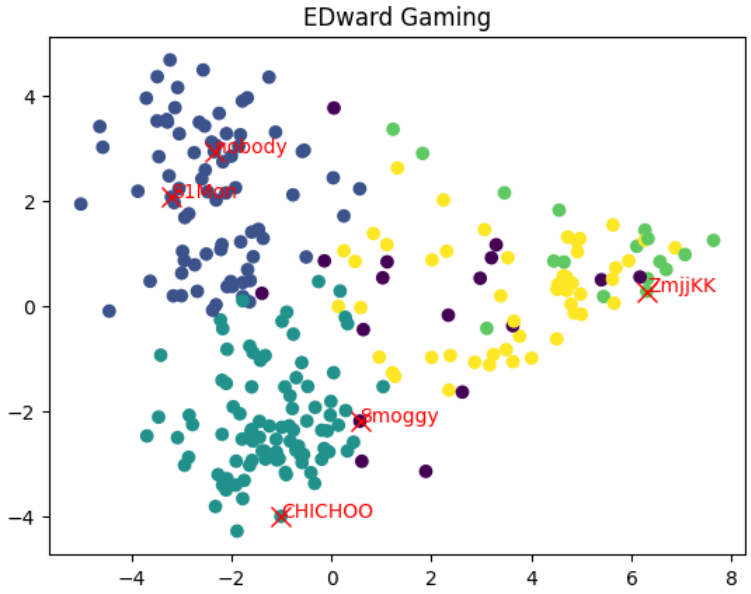
- GEKKO_PICK_RATE - Percent rounds played as Gekko
- HARBOR_PICK_RATE - Percent rounds played as Harbor
- ISO_PICK_RATE - Percent rounds played as Iso
- JETT_PICK_RATE - Percent rounds played as Jett
- KAYO_PICK_RATE - Percent rounds played as Kayo
- KILLJOY_PICK_RATE - Percent rounds played as Killjoy
- KILLS_ARES - Likelihood of getting kills with an Ares
- KILLS_BUCKY - Likelihood of getting kills with a Bucky
- KILLS_BULLDOG - Likelihood of getting kills with a Bulldog
- KILLS_CLASSIC - Likelihood of getting kills with a Classic
- KILLS_FRENZY - Likelihood of getting kills with a Frenzy
- KILLS_GHOST - Likelihood of getting kills with a Ghost
- KILLS_GUARDIAN - Likelihood of getting kills with a Gaurdian
- KILLS_JUDGE - Likelihood of getting kills with a Judge
- KILLS_MARSHAL - Likelihood of getting kills with a Marshal
- KILLS_MELEE - Likelihood of getting kills with a Knife
- KILLS_ODIN - Likelihood of getting kills with an Odin
- KILLS_OPERATOR - Likelihood of getting kills with an Operator
- KILLS_OUTLAW - Likelihood of getting kills with an Outlaw
- KILLS_PHANTOM - Likelihood of getting kills with a Phantom
- KILLS_SHERIFF - Likelihood of getting kills with a Sheriff
- KILLS_SHORTY - Likelihood of getting kills with a Shorty
- KILLS_SPECTRE - Likelihood of getting kills with a Spectre
- KILLS_STINGER - Likelihood of getting kills with a Stinger
- KILLS_VANDAL - Likelihood of getting kills with a Vandal
- NEON_PICK_RATE - Percent rounds played as Neon
- OMEN_PICK_RATE - Percent rounds played as Omen
- PHOENIX_PICK_RATE - Percent rounds played as Phoenix
- RAZE_PICK_RATE - Percent rounds played as Raze
- REYNA_PICK_RATE - Percent rounds played as Reyna
- ROUND_NUMBER - Average rounds played per game
- SAGE_PICK_RATE - Percent rounds played as Sage
- SIDE - Tenancy towards attack(positive)/defense(negative)
- SKYE_PICK_RATE - Percent rounds played as Skye
- SOVA_PICK_RATE - Percent rounds played as Sova

- SPIKE_CARRY_PERCENT - Average duration of round carrying spike
- SPIKE_PLANT - Likelihood to plant spike in a given round
- TIME_ALIVE - Average time alive in a round
- VIPER_PICK_RATE - Percent rounds played as Viper
- VYSE_PICK_RATE - Percent rounds played as Vyse
- YORU_PICK_RATE - Percent rounds played as Yoru

After ingesting every game and constructing round statistics for each player, the individual round statistics can be combined to obtain a player's aggregate $\vec{p}$. Augmenting these vectors together gives us a matrix P that we can data mine.

The components of P can vary wildly with scale and needs to be hit with the normalization stick to stabilize the dataset. Since this is data generated by humans its safe to say these statistics follow a normal distribution. Therefore it is safe to hit the vector with a power transformation and since we have some negative values here, the natural choice is a Yeo-Johnson transformation.

Now with clean and normalized data, we can run BGMM and expect decent results. Then by doing a quick PCA and plotting the first two principle components we can visualize the clusters.



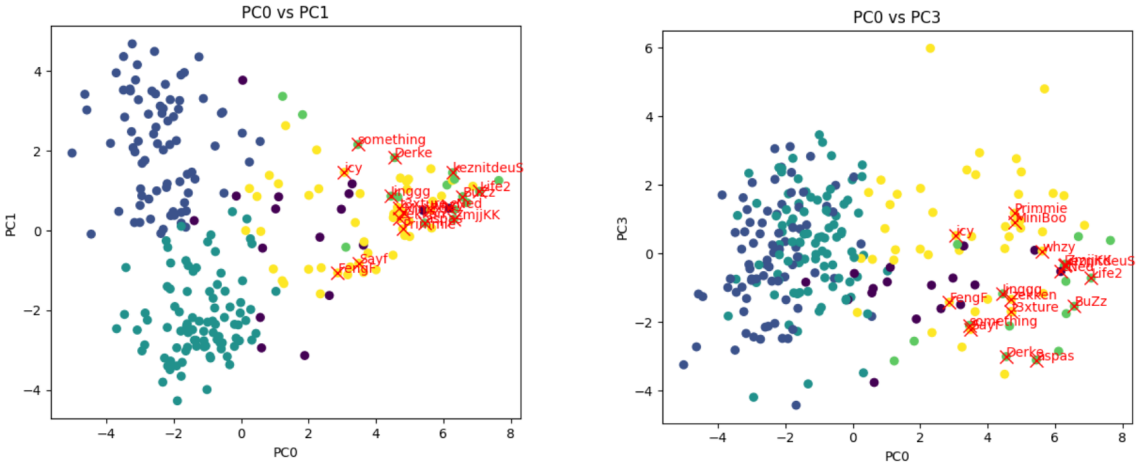
Then by overlaying the players for each role on top of this chart, we can start to look for patterns to filter and rank all players participating in VCT International by role and by performance.

2.1.2 Principle Component Analysis

When looking at known players and the clusters it quickly becomes apparent that the clusters by themselves are not good enough to identify roles and certainly not good enough to rank something as ethereal as performance. This is where the PCA comes in. By studying patterns in player spectral decompositions we can gain additional insight into behavioral patterns of roles while remaining agnostic to personal bias. We can then use the patterns we find and introduce some intuition of what the ranking of VCT Champion players are to define a generalized ranking algorithm that can be applied to the results of the PCA filtering. Once the ranking and player pools for each role and region are decided, all we do is another quick normalization and obtain probability distribution functions (PDFs) for reach role for each region. Looking at the singular values, we determined that 90% of the variance in P is captured by the first 30 principle components (PC_{1-30}) and will be limiting our analysis to those 30. So now its time to look at a bunch of graphs.

2.1.2.1 Duelist

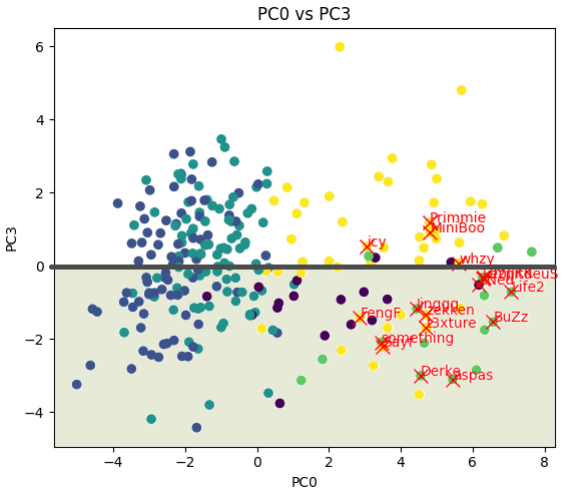
For dualists, the clustering worked quite well. As you can see from the image, they all have collected at the right tip of the triangle, in the green and yellow clusters. Further analysis into the clusters show that the better duelists are in the yellow cluster so we only select players in the yellow cluster. Looking at PC_0 vs PC_3 we see direct correlation between PC_0 and inverse correlation between duelist quality on PC_3.



Further most good duelists have a negative PC_3 coefficient. So we filter to only select from the yellow cluster with players having negative PC_3. We then mix together the PC_0 and PC_3 values to come up with a "dualist score":

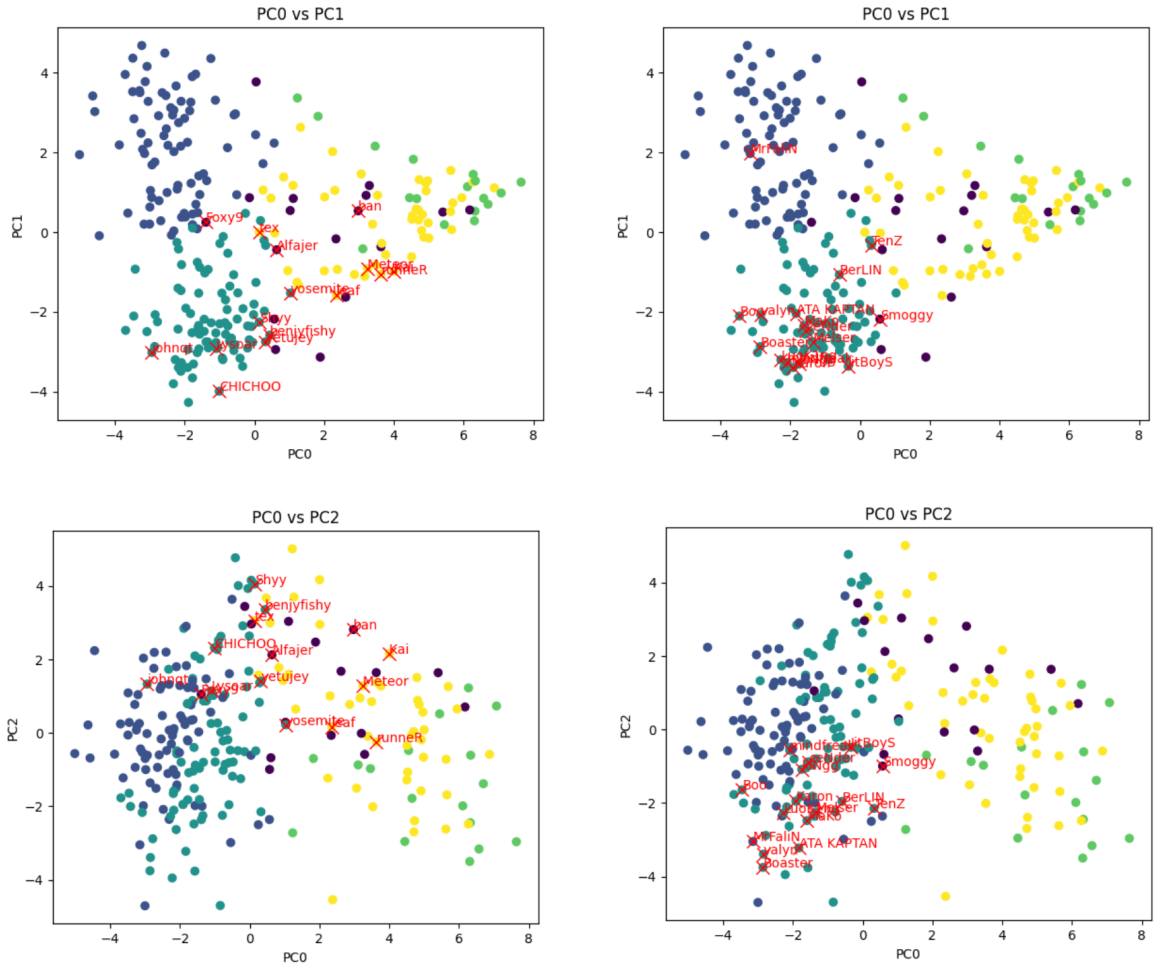
$$S_{duelist} = 0.6 * PC_0 - 0.3 * PC_3$$

and sort the based on their $S_{dualist}$. After doing this we noticed that the good dualists tended to fall in the middle of this list so we used a neat Monty Carlo trick to overlay a normal distribution centered at the middle of the list and then sampled the distribution for a sufficiently large number of times, counting the frequencies each player got selected. We then re-sorted the list based of number of times sampled and converted the frequencies into a pdf.

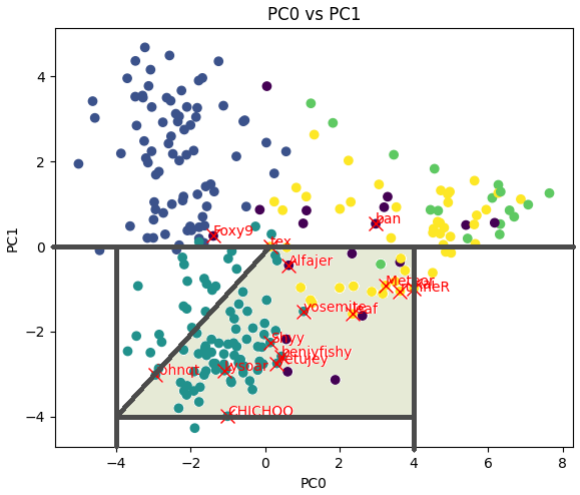


2.1.2.2 Sentinels and Controllers

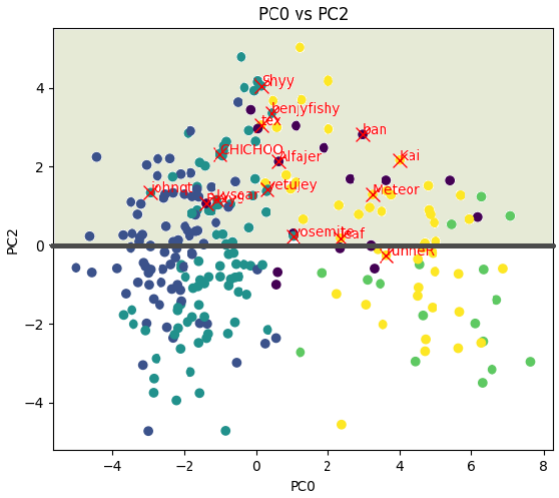
With the sentinels and initiators we see the true power of the PCA.
From the two plots you can clearly see that both sentinels and initiators fall under the teal group and at first glance there is no way to differentiate them. But when we look at PC_2 suddenly we see clearly that sentinels have a positive PC_2 while controllers have a negative PC_2. The sentinels group,



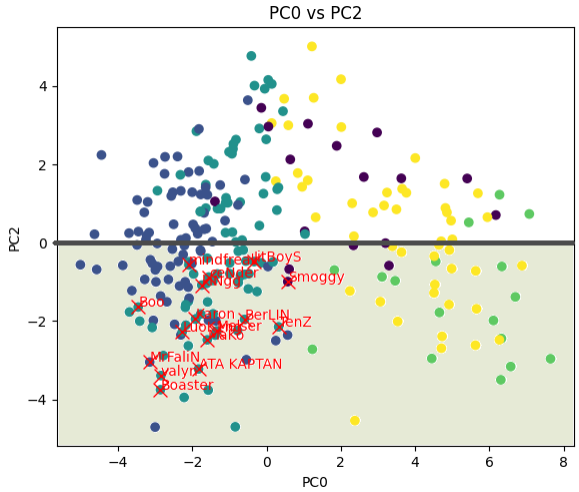
however needed a little more filtering work as it was capturing a lot of players who were very much not sentinels. We noticed that on the PC_0 vs PC_1 graph sentinels always fell under the $y = x$ line, so adding a simple $PC_1 < PC_2$ in-equality proved sufficient.



To rank the sentinels, we noticed good players aggregating around $PC_{23} = -0.75$ on so we ranked them based of distance from that line and created pdfs based of the distance. This pattern ended up cropping up a lot so we decided to refer to it as "component distance ranking".

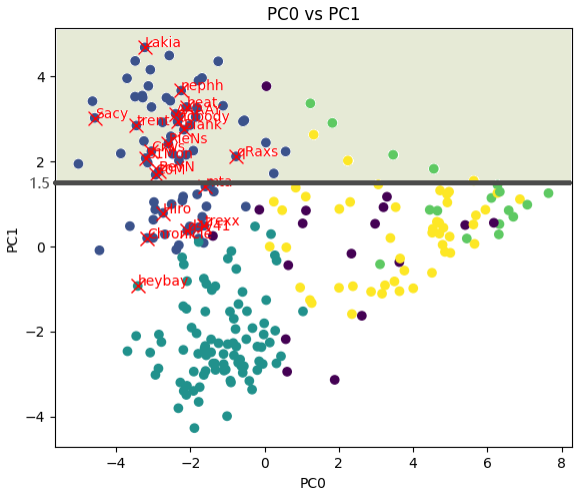


To rank the controllers, we saw clustering around $PC_{12} = -0.25$ and just did component distance ranking again.



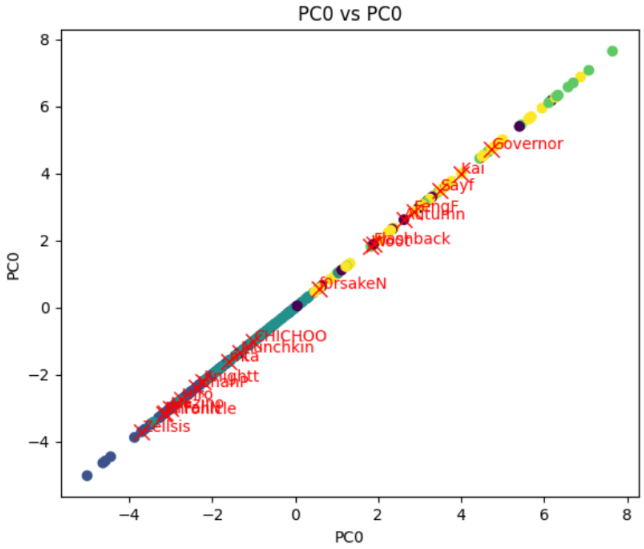
2.1.2.3 Initiators

Initiators is another clustering success case, showing clear aggregation in the blue cluster so we set a performance cutoff at $PC_1 = 1.5$. Furthermore, we saw clustering around $PC_3 = -1.25$ and again used component distance ranking.

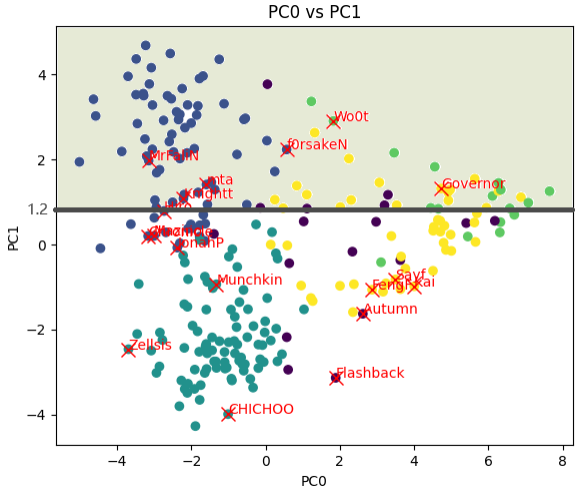


2.1.2.4 Flex

Flex is a little more interesting. Across all clusters there was not coherent pattern, which makes sense given these are flexible players by nature. One interesting thing that did crop up was our ability to detect an initiator-duelist flex and a control-sentinel flex by looking at just the PC_0 graph.

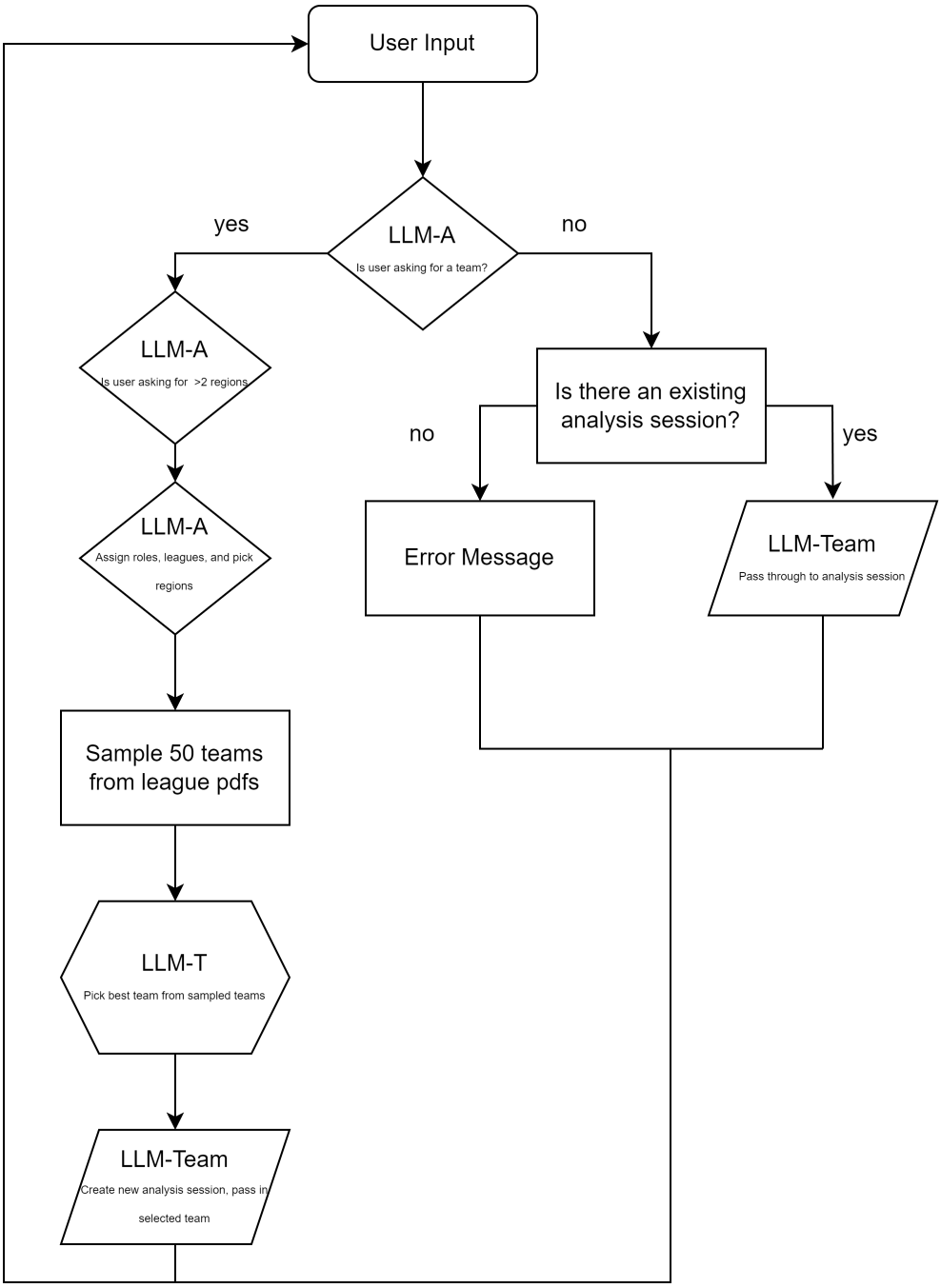


Luckily during our data processing we also kept track of player role pick rate, removing it before doing the unsupervised learning. Adding those stats back in, for each player, we pick the role with the highest pick rate, then sort it in descending order to get the players with the lowest max pick rate, giving an indication of role flexibility. For the flex players we noticed that those with very positive PC_1 values tended to perform better so we created a somewhat arbitrary cutoff at $PC_1 = 1.5$. Interestingly when this cut off is applied to game changers data, no players show up, so we had to lower it to $PC_1 = 1.2$ to get a player pool. We then noticed aggregation of good flex players around $PC_{23} = -0.9$ and again used component distance ranking.



2.2 Application

Now is where the fun stuff happens. When user input comes in requesting a new team, we intercept that message and sample 50 teams, comprising of one player from each role from regions according to the user's request. Then we pass those teams with the respective $\vec{p}$ vectors into a prompt engineered one shot LLM (LLM-T) and have it return the best team. Then we pass this team to another prompt engineered LLM (LLM-Team) to create the final analysis for the end user and a chat session for the user to ask additional questions about the team. This method relies heavily on the LLM base models to have good contextual awareness of what Valorant is, what a Valorant game involves, and what the components of our vectors mean. Thus we pick Claude Sonnet as the basis for LLM-T and LLM-Team to leverage its size and intelligence. In order to make the user experience more seamless, we need to do classification on every user input to see if it is requesting a new team. We can once again turn to the power of an LLM (LLM-A) rather than training up a new classification network. If we detect a new team request we can reset the LLM-Team context and generate a new team. Otherwise we can pass the user input to LLM-Team to give the illusion of a direct chat session. After checking whether the prompt is requesting a new team, we also prompt LLM A to see whether the request is asking for a team with 3 or more different regions. If this is the case, we can ensure that we select a diverse group of players. Otherwise, we try and stick to the VCT import rules of only one imported player. Both of these classifications need to be fast so we decided to use Claude Haiku to save on cost and minimize latency. Finally we need to extract what leagues the user would like us to draw from. Since this is also a simple natural language analysis job, we can re-use LLM-A with a different primer to return an optimal role-league mapping. The final flow of the application ends up looking like this:



3 Implementation

The implementation of this system starts in Google Collab, where we wrote initial scripts to analyze small sets of the data and develop a proof of concept. We then take the notebook and bring it over to AWS Glue, adapting the scripts to use pyspark so as to scale to the hundreds of gigabytes of json data provided by the competition. We then save the processed $\vec{p}$ data as a parquet on an S3 instance.

*Side note here: This is how we learned the hard way that Jupyter Notebooks + AWS Glue = ballooning server costs and that running scripts is MUCH more cost efficient. Even with the notebooks already written we were able to rack up $300 of fees by debugging inside of active notebook sessions before we noticed.

Once we have the parquets, we can bring the analysis back into Google Collab to train models, study the principle components, and generate the pdfs. Finally, using Streamlit we implement the user interaction logic tree and polish up a nice UI to improve presentation.

4 Known Issues

Due to time constraints and no easy access to individual player original nationality (rather than the home team region provided in the dataset), we just used the players home team region as their region. This is sometimes inaccurate as an imported player from Korea to America, for example, would show up as American. With a little more time to scrape the data, it would be an easy add given our architecture.

We were running into issues finding enough players that passed the various filters when regions were very specific, like in VCT Challengers and Game Changers. To remedy this we had to combine sub-regions back into their parent regions like in VCT International. This combined with the fact that we could not find import rules for VCT Challengers and Game Changers means that the generated pure teams may not be entirely legal.

For some reason AWS Bedrock has decided to limit our request quotas to be quite low. In order to fail gracefully from a trottle exception we go to sleep for a exponentially longer times to let the rate reset and notify the user that the chat bot needs to slow down for a bit. Generally this shouldn't effect use too much since we do very few API calls to the LLM models but is a fail state that should be handled.