Optimising Recommendation Systems

Sai Prakash
Dept of Computer Science
College of Science, SJSU
San Jose, USA
sai.prakash01@sjsu.edu

Karan Jain
Dept of Computer Science
College of Science, SJSU
San Jose, USA
karan.jain@sjsu.edu

Abstract - **There exist many diverse types of recommendation systems, based on the features present in any given system. Reinforcement Learning, while effective, has not been used as much in such recommendation systems, however there are certainly instances where reinforcement learning can help in the optimization of recommendation systems. We propose the use of reinforcement learning in recommendation systems to improve the reward/clickthrough rate of specific products. We use 2 different ecommerce-based environments, RecoGym and Amazon Fashion Reviews; build agents over both environments and measure the effectiveness of such systems. Additionally, we deployed an agent - environment decoupling system for the latter. We see that reinforcement learning proves to be an effective way to improve clickthrough rates over regular machine learning. We managed to improve clickthrough rates in RecoGym simulations. Additionally, we propose a scalable distributed data pipeline, that we have utilized for Deep Reinforcement Learning Recommendation Systems using the Amazon Fashion Review Dataset.**

Keywords—recommender systems, machine learning, reinforcement learning

I. INTRODUCTION

Product recommendations on e-commerce sites are a key driver of revenue generation, and our project set out to optimize these systems. To facilitate recommendations, several recommendation systems exist out there that use classical Machine Learning (ML) pattern prediction and association rules. We rephrase the recommendations system as a reward - action environment and apply Reinforcement Learning (RL) because framing it as a RL problem, we can identify optimal recommendation actions that directly and indirectly increase the revenue. Additionally, we address the practical challenges faced while deploying these systems in real-time environments, by employing a decoupled services system. This separates the agent and the environment and ensures computational efficiency and cost-effectiveness. We successfully implemented and evaluated several approaches, including traditional ML and RL models, to enhance personalized advertisement recommendations. Our results demonstrated the significant potential of Reinforcement Learning in transforming the recommendation problem into an optimization task and a de-coupled system that can handle interactions between the agent and the environment.

II. BACKGROUND

Existing research on Deep Reinforcement Learning (DRL) based Recommendation Systems involves the use of supervised ML techniques, to determine the embedded features of a given dataset to extrapolate important inferences. [1] delves into the limitations of traditional supervised ML models in recommendation system tasks. The authors further explain why the task of recommendation owes itself well to RL and how it can be formulated as a reinforcement learning problem. The focus in this work is modeling user and item interaction using an end-to-end framework created, that incorporates RL for recommendation and uses supervised ML to improve the representation of items. The authors claim that the DRL value based, and policy-based approaches outperform traditional interactive recommendation systems, however these claims cannot be generalized or applied to recommendation systems with limited amount of user or feature data. While this system does yield great benefits in accuracy of prediction, the computation cost and latency of such systems may make them infeasible for real-time applications. Such recommendations also do not perform well when there is limited feature data and when cold starts for recommendations are required.

A significant amount of focus and research has gone into adaptive machine learning and how it can be helpful when deployed in the field of customer success [2]. Assessing customer feedback and prioritizing responses based on the sentiment analysis of feedback received can significantly boost response times and improve customer satisfaction and overall User experience. Adaptive machine learning is an especially interesting area of research. For customer-facing systems, businesses often require machine learning models capable of adapting dynamically to changes and ad hoc customer requests. RL emerges as an important approach to address these needs.

Early recommendation systems usually utilized some form of supervised machine learning. [3] introduced a different RL approach to recommendation systems, moreover it introduced a way to test recommendation systems in a simulated environment. The authors have also made this simulation software available to us in the form of the RecoGym package in python. The authors have conceptualized a "recommendation game", in which we need to develop agents to optimize the reward obtained by the agent. This recommendation simulation is something that is especially useful for simulating and testing agents in e-commerce platforms.

Feed-streaming recommender systems need engagement rewards and increased desire to stay for the real time user. Traditionally, these systems often optimize their performance for short term metrics like click-through rate or immediate engagement and lose focus on delayed metric like long term satisfaction or retention. They also do not scale up well enough. Other off-policy learning methods face issues of high variance with increased item candidates. RL is quite appropriate for such a problem, as it considers long-term rewards for optimizing user engagement. recommender system learns to make the future more rewarding as per the users' interaction. Applying RL to such a problem still faces challenges such as versatile user behavior and ineffective off-policy learning. To mitigate this, the [4] introduced an RL model for feed streaming recommendations combined with Deep Q-Network (DQN) with hierarchical long short-term memory and policy gradient methods. The model also mitigates the deadly triad problem, which introduces instability and divergence. The results show success by outperforming the current state-of-the-art methods.

User profiling is the basis of personalized recommendation systems. It helps discover knowledge graphs from heterogeneous sources, on a basis to the user-item interactions. Usual approaches like neighborhood collaborative filtering make rating predictions but suffer from data sparsity and missed node affiliations. It is better to view this problem as a link prediction, where there is a link between user node and item nodes or interactions. Unlike traditional methods, DRL can also optimize for different long-term goals such as profit or user engagement. However, it suffers challenges due to the arbitrary nature of user interaction and limited training data to train any learning policy. [5] formulates the user profiling process as a Markov Decision Process (MDP) and then sets up an RL agent to solve the MDP. Iteratively, the RL agent is set up to find a quality set of nodes and relations based on both the expert and data-specific domains. The proposed RL agent is trained and tested on three datasets. It generates longer and interesting meta paths compared to usual approaches and is much more scalable vs the state-of-the-art recommendation systems.

Unlike traditional recommender systems, interactive recommender systems (IRS) make multistep predictions and update themselves at each step. [6] addresses the challenge of creating more effective IRS by incorporating additional contextual information. Traditional RL methods like multi-armed bandit for IRS often struggle with dynamic environments, limiting their performance. Newer approaches like DRL e.g. DQN and Deep Deterministic Policy Gradient show promising results but suffer from sample efficiency as limited interactions cannot train DRL effectively. The authors propose a new framework that integrates knowledge graphs (KG) into reinforcement learning, i.e. Knowledge Graph-enhanced Q-learning (KGQR). This involves leveraging KG to enrich item and user state representations and guide candidate item selection. The integration of knowledge graphs into the reinforcement learning process significantly improves the performance of interactive recommender systems. The proposed KGQR framework demonstrates superior results compared to existing state-of-the-art methods in accordance with increased sample efficiency.

III. Datasets And Environment

Recommendation systems can come in many shapes and forms. There are different scenarios that will require different types of recommendation systems. Companies spend large amounts of resources to improve their recommendation systems to squeeze an improvement of even 1%. So, while there are different scenarios we could explore, this project explores 2 different scenarios, where we would ideally like to introduce RL recommendation systems:

- Recommendation Systems where we may not have any data of the user preference, knowledge about the products, or any additional info to support how recommendations must be made. These are cold-start recommendation systems where we need to derive each product's apparent popularity given the clickthrough rate by trial and error.
- Systems where we have additional data, not only clickthrough rate that can support inferences that are made by the recommendation system. In these inference driven systems, we usually have some additional metadata that can allow us to drive recommendations based on user interaction.

A. RecoGym

RecoGym[7] is an OpenAI gym-based environment that simulates user traffic patterns and interactions with e-commerce recommendations. This allowed us to rigorously test and refine our algorithms under simulated conditions. The reason we chose to use the RecoGym environment is that often we do not have additional data regarding products and services or user bias. In such scenarios we need a method to ideally maximize the clickthrough rate while under the least number of iterations. We compared various approaches, demonstrating how Reinforcement Learning outshines traditional classification models in adaptability and effectiveness. In our tests we ran the simulations with 10 products and 5000 users.

In RecoGym there are different kinds of simulated experiences and episodes. There is an organic session, which can be explained simply as an observation of what the user interacts with, note that in such a session an agent does not provide any recommendations. There are also bandit sessions, in which an agent can interact with the environment, choosing to provide a list of recommended products. If the simulated user decides to click on the product, then the agent receives a reward of +1 and 0 otherwise. In most simulations, users will shift between organic and bandit interactions. However, the organic interactions are returned as observations during a bandit session. Hence an observation may look something like this:

Observation: [{'t': 6, 'u': 0, 'z': 'pageview', 'v': 4}, {'t': 7, 'u': 0, 'z': 'pageview', 'v': 4}, {'t': 8, 'u': 0, 'z': 'pageview', 'v': 4}]

Here t is the timestep, u is the user id, z is the observation type. In our case we will always use 'pageview' so this shall remain the same throughout our simulations and v is the item id of the product. And an action may look like this:

Action: {'t': 1, 'u': 0, 'a': 3, 'ps': 0.1, 'ps-a': array([0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1])}

Where *t* is the time step, *u* is the user id, *a* is the action selected. *ps* is the propensity score or the probability of selecting the action *a*, and *ps-a* is the probability of selecting each of the available actions.

B. Amazon Fashion Review

IV. METHODOLOGY

A. Simulations using Recogym

To understand and measure the performance of recommendation systems that need to improve clickthrough rate while, we used the RecoGym package. We have set the number of users as 5000 and the number of products as 10. So, to check how our models perform we will utilize the A/B Testing method provided in RecoGym.

We have created different bandit models; we have created a simple neural network model as well as a logistic regression bandit model that have been trained on generated simple log data of 5000 users. Once the models are trained using the generated data. To train our models we need to use relevant features first, but in our simulation environment, we have no prior information about user bias, and hence we need to find a way to improve our recommendations without user through likelihood. Therefore, we train our models based on the likelihood of a specific product.

Here, we count the number of times a product has occurred during an organic session and make a list of the number of times each product has occurred. So for example if our simulation is set to have 10 products during log creation there is an organic event, which is then followed by a bandit event, the feature input for the bandit event would look something like this - [3,0,0,0,0,0,0,0,0,0]. This means that in the prior organic event the user has visited the first product 3 times and the other product 0 times. We construct similar features for every bandit event input and feed it to our model to generate an appropriate output.

We have implemented our models on Google Colab. The simple neural network is a 3-layer neural network. The neural network and the logistic regression model both take in the input features and output the item that is chosen for recommendation.

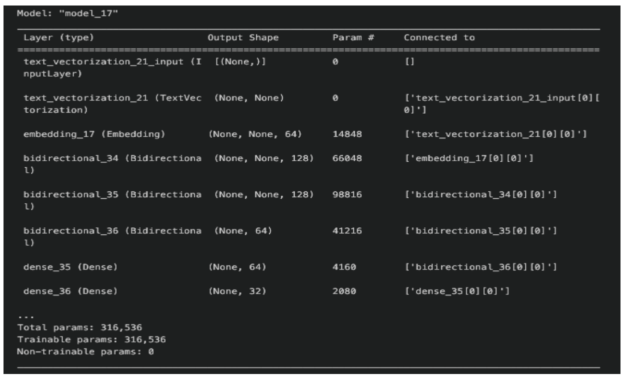
B. Inferencing Model On Amazon Fashion Environment

To work with a more realistic dataset and to understand how such recommendation systems can be created in such a model we decided to create a simple environment based on the Amazon Fashion Review dataset. The main focus of this part of the project is to highlight our decoupled environment and highlight how such a data pipeline will be useful when deploying such models at scale.

Data Preparation:

We have utilized the Amazon Fashion Review dataset to showcase how inference driven models could work in the context of Reinforcement Learning , the dataset contains 828,699 rows which are reviews of various amazon fashion products, while there is a significant amount of review data, a lot of the data is not required by us and is not. The dataset contains a variety of products with mixed reviews, ratings and number of reviews. The dataset contains information about the product id, reviewer id, review score, review text, summary and timestamp. We care about the review text content and the product metadata which gives us characteristic information about the product itself. The state over here represents the current product, and the future states is the different products that the user may go to, we calculate this in the environment by checking if a particular user has reviewed a similar product after reviewing the current product. The action fed to the environment denotes whether a particular product is recommended or not. If a product is recommended, the environment emits a reward of 1, else the environment emits a reward of 0.

- Dataset Ingestion: As the review dataset contains little over 880k rows, we need to reduce the size of dataset to a feasible size for the neural network to learn. To do this we first create a data frame over the data for easier manipulation. We create a list of products and reviewers separately.
- Since we want to phrase this problem as a reinforcement learning task instead of a supervised learning task, we needed reliable reviews to base recommendation, hence we further cleaned the dataset to retrieve reliable reviews form reviewers who had at least 10 product purchases.
- Grouping: We group the reviews over products, reviewers and review time. Now we have a corpus of about 1.6k reviews grouped and cleaned.
- States: To depict the characteristics and user of each product, we develop product states and store sizes, colors and reviews onto these states. We iterate over each product and extract these details.
- Metadata: While size and color are implicitly proper nouns and thereby features, we need to extract the features from the reviews explicitly, and we do this by using the *spacy* python library to extract nouns and proper nouns from the reviews. This goes into the states' metadata for each product. We collate the metadata for each product and remove any unavailable.
- Reward: If the Agent recommends a product that is present in future possible states, the agent gets a reward of +1 else, else the agent gets a reward of 0.



```
Model: "model_17"
________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
================================================================================================
 text_vectorization_21_input (I  [(None,)]           0           []
 nputLayer)

 text_vectorization_21 (TextVec  (None, None)        0           ['text_vectorization_21_input[0][
 torization)                                                      0]']

 embedding_17 (Embedding)       (None, None, 64)     14848       ['text_vectorization_21[0][0]']

 bidirectional_34 (Bidirectiona  (None, None, 128)   66048       ['embedding_17[0][0]']
 l)

 bidirectional_35 (Bidirectiona  (None, None, 128)   98816       ['bidirectional_34[0][0]']
 l)

 bidirectional_36 (Bidirectiona  (None, 64)          41216       ['bidirectional_35[0][0]']
 l)

 dense_35 (Dense)               (None, 64)           4160        ['bidirectional_36[0][0]']

 dense_36 (Dense)               (None, 32)           2080        ['dense_35[0][0]']

...
Total params: 316,536
Trainable params: 316,536
Non-trainable params: 0
________________________________________________________________________________________________
```

Fig 1: Our DQN Architecture

We run our model for a maximum of 50 steps per episode. We can see that after training the first 50-episode our model reaches optimal performance. The model takes in the current step at each iteration time step, adds the state information to the replay buffer and then outputs an action. This action is the recommendation of the next product. However, this is just a sample custom environment built for us to test our data pipeline and highlight the decoupled nature of training using this data pipeline. The environment itself is very basic and may not actually be a proper model of user interaction or website interaction.

Data Pipeline

The *Kafka*, *Cassandra* and *Airflow* servers are hosted on a *Docker* stack for the purpose of simulation. There is 1 *ZooKeeper* that handles them all. The zoo initially contained 3 *Kafka* brokers, 1 *Schema-Registry* service and 1 *Kafka Connect* service. Additionally, we added a *Kafka UI* service as a visual tool to view the topics and consumers, and a *REST Proxy* service to able to handle GET and POST queries from producers and consumers.

We then use *ngrok* to create a reverse proxy server which facilitates communication between the Rest Proxy server and the producer/consumer over internet. This truly de-couples the agent and the environment as they do not interact directly within themselves but rather via message queuing on *Kafka*.

We create different topics for actions and state i.e. *fashion_action* and *fashion_state*. The agent writes its action back to the topic *fashion_action* and consumes the state of the environment from the *fashion_state*. The environment consumes the actions by the agent from the *fashion_state* and then writes back its state to the *fashion_state*. The usual Time To Live (TTL) for any message is about 2 days, but this can be changed based on our preferences. In our simulation the agent and the environment interact with each other via the Kafka message queue. This not only creates a robust and fault tolerant system but also ensures that data can be read simultaneously by multiple concurrent users/agents. Additionally, Apache Kafka is known for achieving high throughput, which can be beneficial for such systems.

During the simulation the agent and environment run on the python environment in Google Colab. To send a message to any topic they send a GET query to the reverse proxy *ngrok* server, which is forwarded to the exposed the REST Proxy service. The REST Proxy service then forwards to the Kafka broker(s) and the message is updated onto the topic. To recieve a message from the topic the agent or the environment has to subscribe to the topic by creating a consumer group and then subsequently a consumer. Then it polls for 3 retries each time and when received it performs the next step of the simulation.

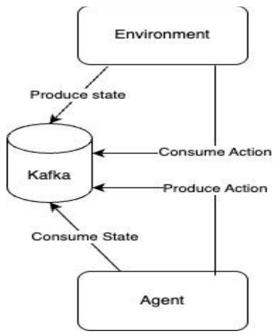


Fig 2: Agent and Environment Interactions

At the end of day, the data is additionally consumed back to a permanent data store i.e. the *Cassandra* using a pipeline created in *Airflow*. This ensures that the actions and state are preserved, and the *Kafka* brokers are not overwhelmed with storing the data forever.

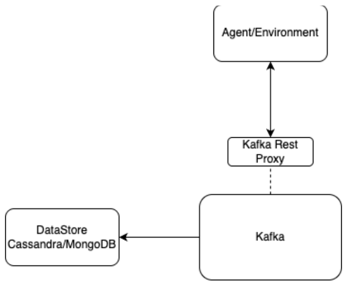


Fig 3: Message write back to permanent data store.

V. Results

A. RecoGym Simulations

Given our extensive use of RecoGym we wanted to improve our model performance and try different variations of bandit simulations. In this paper based on the simulations tried when setting the number of users to 5000, number of items to 10 we achieved the following results:

Table 1 : Recogym Simulation Results

Model Name	Average Click Through Rate
Neural Network	0.119
Organic Count	0.162
Log Likelihood	0.186

We find that the neural network model that we implemented seems to perform the most poorly. This is most probably due to the model grossly overfitting, with very limited amount of feature data. We can also see that the log-likelihood logistic regression model outperforms the organic agent count model reaching an average clickthrough rate of around 0.183. There are further improvements to be made here to further improve the clickthrough rate, which would require more time and effort. We must note that there every 0.01 improvement in clickthrough rate plays a very important role for e-commerce websites, or other platforms where clicks can come in the order of millions if not billions.

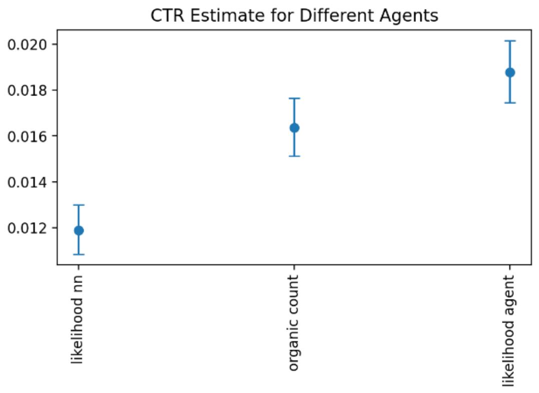


Fig 4: Improved clickthrough rate by agent

As we would expect, it seems that likelihood models seem to perform the best when there is limited data available. However, it is possible that during cold starts, we will not know the exact likelihood of specific items, we can approximate this based on past data, but there is a chance that given the nature of the bandit problem that we do not explore actions which may in fact have higher popularity. This is something that should be explored further.

B. *Amazon Fashion Review Simulations*

On training our model for 50 episodes, we can see that this is the result we achieve.

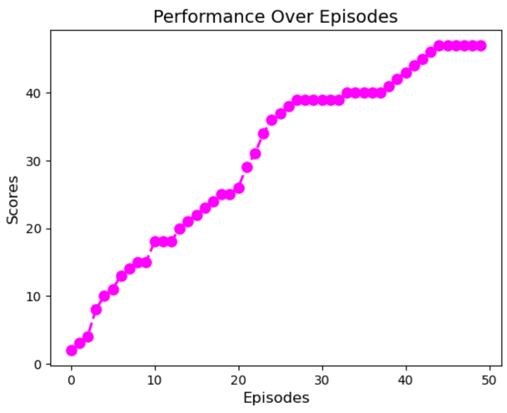
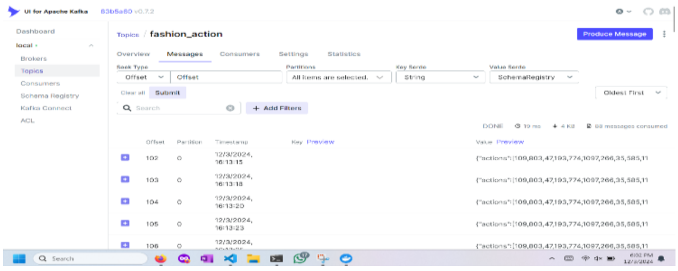


Fig 5: Increasing scores per episode

This result can be improved significantly if we simply increase the model size and the dataset size. However, the goal of this part of the project is not to show different models that can be used to solve this task but rather the data pipeline where we can decouple the environment from the agent using *Kafka* as a message broker.



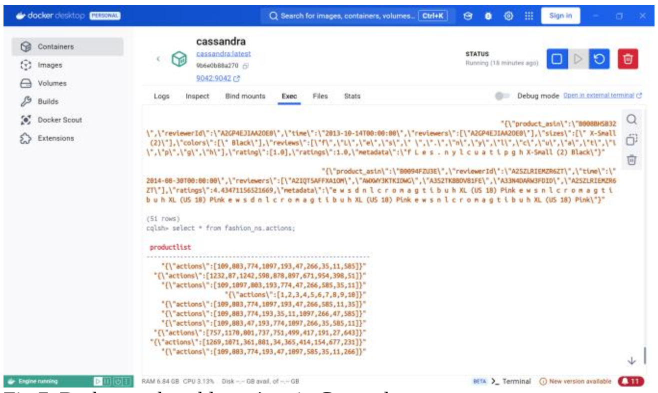
```
Seek Type: Offset   Offset   Partitions: All items are selected.   Key Serde: String   Value Serde: SchemaRegistry

DONE   19 ms   4 KB   68 messages consumed

Offset  Partition  Timestamp            Key Preview   Value Preview
102     0          12/3/2024, 16:13:15                {"actions":[109,803,47,193,774,1097,266,35,585,11
103     0          12/3/2024, 16:13:18                {"actions":[109,803,47,193,774,1097,266,35,585,11
104     0          12/3/2024, 16:13:20                {"actions":[109,803,47,193,774,1097,266,35,585,11
105     0          12/3/2024, 16:13:23                {"actions":[109,803,47,193,774,1097,266,35,585,11
106     0          12/3/2024,                         {"actions":[109,803,47,193,774,1097,266,35,585,11
```

Fig 6: A view of the Kafka topic *fashion_action*

In the simulation we have created we have tested the setup for one agent but by using Kafka as a message broker we can have multiple different agents subscribe and produce messages to the topics. It is due to the decoupling nature of the system that the number of agents can be scaled up or down if needed.



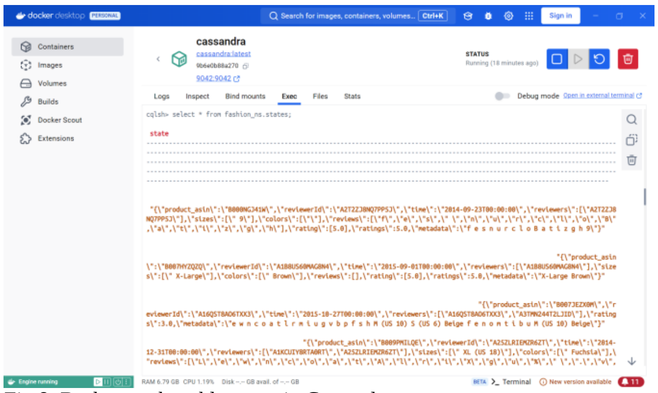
```
                                                                         "{\"product_asin\":\"B0008HSB32
\",\"reviewerId\":\"A2GP4EJIAA2OE0\",\"time\":\"2013-10-14T00:00:00\",\"reviewers\":[\"A2GP4EJIAA2OE0\"],\"sizes\":[\" X-Small
 (2)\"],\"colors\":[\" Black\"],\"reviews\":[\"f\",\"L\",\"e\",\"s\",\" \",\".\",\"n\",\"y\",\"l\",\"c\",\"u\",\"a\",\"t\",\"i
\",\"p\",\"g\",\"h\"],\"rating\":[1.0],\"ratings\":1.0,\"metadata\":\"f L e s . n y l c u a t i p g h X-Small (2) Black\"}"

                                         "{\"product_asin\":\"B0094FZU3E\",\"reviewerId\":\"A2SZLRIEMZR6ZT\",\"time\":\"
2014-08-30T00:00:00\",\"reviewers\":[\"A2IQT5AFFXA1OM\",\"AWDWY3KTKIDWG\",\"A3S2TKBBDV81FE\",\"A33N4DARW3FDID\",\"A2SZLRIEMZR6
ZT\"],\"ratings\":4.434711565216 69,\"metadata\":\"e w s d n l c r o m a g t i b u h XL (US 18) Pink e w s n l c r o m a g t i
b u h XL (US 18) Pink e w s d n l c r o m a g t i b u h XL (US 18) Pink e w s n l c r o m a g t i b u h XL (US 18) Pink\"}"

(51 rows)
cqlsh> select * from fashion_ns.actions;

 productlist
----------------------------------------------------------
   "{\"actions\":[109,803,774,1097,193,47,266,35,11,585]}"
 "{\"actions\":[1232,87,1242,598,878,897,671,954,398,51]}"
   "{\"actions\":[109,1097,803,193,774,47,266,585,35,11]}"
                    "{\"actions\":[1,2,3,4,5,6,7,8,9,10]}"
   "{\"actions\":[109,803,774,1097,193,47,266,585,11,35]}"
   "{\"actions\":[109,803,774,193,35,11,1097,266,47,585]}"
   "{\"actions\":[109,803,47,193,774,1097,266,35,585,11]}"
  "{\"actions\":[757,1170,801,737,751,499,417,191,27,643]}"
 "{\"actions\":[1269,1071,361,881,34,365,414,154,677,231]}"
   "{\"actions\":[109,803,774,193,47,1097,585,35,11,266]}"
```

Fig 7: Backup to the table *actions* in Cassandra store



```
cqlsh> select * from fashion_ns.states;

 state
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------


 "{\"product_asin\":\"B000NGJ41W\",\"reviewerId\":\"A2T2ZJ8NQ7PP5J\",\"time\":\"2014-09-23T00:00:00\",\"reviewers\":[\"A2T2ZJ8
NQ7PP5J\"],\"sizes\":[\" 9\"],\"colors\":[\"\"],\"reviews\":[\"f\",\"e\",\"s\",\" \",\"n\",\"u\",\"r\",\"c\",\"l\",\"o\",\"B\"
,\"a\",\"t\",\"i\",\"z\",\"g\",\"h\"],\"rating\":[5.0],\"ratings\":5.0,\"metadata\":\"f e s n u r c l o B a t i z g h 9\"}"


                                                                                                            "{\"product_asin
\":\"B007HYZQZQ\",\"reviewerId\":\"A1B8US60MAG8N4\",\"time\":\"2015-09-01T00:00:00\",\"reviewers\":[\"A1B8US60MAG8N4\"],\"size
s\":[\" X-Large\"],\"colors\":[\" Brown\"],\"reviews\":[],\"rating\":[5.0],\"ratings\":5.0,\"metadata\":\"X-Large Brown\"}"


                                                                                    "{\"product_asin\":\"B007JEZX0M\",\"r
eviewerId\":\"A16QST8AO6TXX3\",\"time\":\"2015-10-27T00:00:00\",\"reviewers\":[\"A16QST8AO6TXX3\",\"A3TMN244T2LJID\"],\"rating
s\":3.0,\"metadata\":\"e w n c o a t l r m i u g v b p f s h M (US 10) S (US 6) Beige f e n o m t i b u M (US 10) Beige\"}"

                                           "{\"product_asin\":\"B009PMILQE\",\"reviewerId\":\"A2SZLRIEMZR6ZT\",\"time\":\"2014-
12-31T00:00:00\",\"reviewers\":[\"A1KCUIYBRTA0RT\",\"A2SZLRIEMZR6ZT\"],\"sizes\":[\" XL (US 18)\"],\"colors\":[\" Fuchsia\"],\
"reviews\":[\"L\",\"e\",\"w\",\"n\",\"c\",\"o\",\"a\",\"t\",\"A\",\"l\",\"r\",\"i\",\"X\",\"g\",\"u\",\"%\",\" \",\".\",\"v\",
```

Fig 8: Backup to the table states in Cassandra store

We have additionally backed up the messages on both the topics back to the Cassandra tables. We orchestrated this using an Airflow DAG that is designed to run daily.

VI. Future Work

The area of recommendation systems is vast. There are different areas of research that future work can focus on to improve recommendation systems as a whole. We mainly looked at cold start, systems where there was limited initial information, but with the boom in LLMs, it might be possible to integrate different aspects of LLMs especially for data augmentation to help improve such recommendation systems.

A. *Environment*

While RecoGym is a great simulator for recommendation systems, it is a bit old, we can potentially use AI augmentation models and LLMs to get a richer environment, with relevant data to train more accurate recommendations agents. We can also additionally create more accurate environments from sampling and understanding more user data. Currently the environments offered for the study of recommendation systems are quite limited and often do not contain rich metadata that simulate real world scenarios. Additionally, if we are able to further create viable models for user interaction, we can introduce planning strategies to conventional Machine Learning models to optimize our recommendations.

B. *Agents*

There are 2 types of agents that are being used in our work, Supervised Learning agents and Reinforcement Learning agents. There are more optimizations that need to be added, that will allow agents to make more accurate predictions based on user data, because when we introduce richer data, we can make more optimized predictions based on user preferences.

C. *Systems*

We proposed architecture that efficiently decouples the environment from the agent. The next step to this process is to deploy such architecture on a cloud platform and analyze the efficiency of individual components and services of our system and enhance their capabilities.

We were limited by the amount of memory, processing power and container sizes due to Docker hosting on a personal computer. This can be improved by hosting on a dedicated server. *Kafka* topics can be used better as we were not able to create an effective and exact schema for our topics although the schema registry server existed.

In the real-world scenarios, there may be multiple environments, each for a sector of ecommerce store like fashion, technology and groceries. We would like to extend our work to such an extent and build context-aware intra-environmental interactions.

REFERNCES

[1] F. Liu et al., "End-to-end deep reinforcement learning based recommendation with supervised embedding," Proc. 13th Int. Conf. Web Search and Data Mining, 2020, pp. 384–392.

[2] N. Upadhyaya, "Enhancing real-time customer service through adaptive machine learning," Mach. Learn., vol. 1, no. 5, p. 17, 2024.

[3] D. Rohde et al., "Recogym: A reinforcement learning environment for the problem of product recommendation in online advertising," arXiv Preprint arXiv:1808.00720, 2018.

[4] L. Zou et al., "Reinforcement learning to optimize long-term user engagement in recommender systems," Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min., 2019, pp. 2810–2818.

[5] H. Liang, "DRprofiling: Deep reinforcement user profiling for recommendations in heterogeneous information networks," IEEE Trans. Knowl. Data Eng., vol. 34, no. 4, pp. 1723–1734, 2020.

[6] I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.