TUM Data Innovation Lab

Munich Data Science Institute (MDSI)

Technical University of Munich

&

inovex GmbH

Final report of project:

Graph Learning Based Fashion Recommendations

Authors	Marina Baumgartner, Aanchal Rajesh Chugh, Paraskevi Kiriakidou, Carlos Llano, Tobias Schamel
Mentors	M.Sc. Frauke Beccard, Dr. Lea Petters
Co-Mentor	M.Sc. Cristina Cipriani
Project Lead	Dr. Ricardo Acevedo Cabra
Supervisor	Prof. Dr. Massimo Fornasier

10th February 2023

Acknowledgements

First of all, we would like to thank our Project Sponsor inovex and our inovex Mentors Frauke Beccard, M.Sc., and Dr. Lea Petters, for their support, encouragement and guidance throughout the project.
Thank you very much for your great personal commitment.

We would also like to thank our TUM Co-Mentor Cristina Cipriani, M.Sc., who has always provided us with valuable feedback over the last few months.

We would like to extend our gratitude to our Project Lead Dr. Ricardo Acevedo Cabra and Prof. Dr. Massimo Fornasier for giving us the opportunity to conduct the project at Munich Data Science Institute (MDSI).

We also gratefully acknowledge the computational and data resources provided by the Leibniz Supercomputing Centre.

Abstract

Graph learning has recently become more popular in the field of recommender systems, as many of the problems that recommender systems try to solve can be modeled as a graph. For fashion recommendations, we typically deal with very large and sparse data sets that make it especially challenging to build high-quality personalized recommender systems. Another challenge in fashion is fairness, as we usually deal with sensitive customer data. This project aims to implement a fashion recommender system using graph-based learning to provide personalized recommendations for customers with fairness evaluation. For this purpose, we explore the public data set from H&M, which contains article data (e. g. name, description), customer data (e. g. age, membership), and their transactions. To capture the complex relationships between customer, article, and their associated properties, we designed a graph database and implemented three graph-based recommender approaches: Random Walk as our baseline, a Graph Embedding, and a Graph Neural Network approach. We evaluated and compared these strategies in terms of how relevant and fair the recommendations are. Our results show that different approaches performed best for differing metrics and demonstrate the potential for graph-based recommender systems in the fashion domain.

Contents

1 Introduction

In today's world, especially online, customers constantly have the choice of which content to consume or which products to buy. The ever-growing number of items to choose from makes it increasingly difficult for users to make a good choice in a reasonable amount of time without assistance. Customers can be supported by recommender systems in their decision-making process. However, recommender systems do not only provide a benefit to the user but also to the seller of items to ease sales as well as to improve customer satisfaction and customer loyalty.

1.1 Recommender Systems

Recommender systems are software tools that try to recommend users items they might be interested in. These items can be, among others, products in an online shop, textual content in social networks as well as movies on streaming platforms. Recommender systems make use of data about the user's previous behavior to generate these recommendations. There are two main approaches used to tackle the problem of generating recommendations: collaborative filtering and content-based filtering.
Collaborative filtering, a term that was first introduced in 1992 for a newsletter-mail system, mainly relies on features associated with the user for whom recommendations shall be generated [6]. Systems using collaborative filtering try to find users with a similar history of user-item interactions and suggest items that these users with a similar history also interacted with. Collaborative filtering in an online shop would aim to find a user with a similar purchase history to the customer served and recommend those products that the customer has not bought yet.
Content-based filtering requires an understanding of the items that are recommended. Algorithms using content-based filtering try to recommend similar items to those in the history of user-item interaction of the user for whom recommendations are generated. A user of an online streaming service who mainly watched a certain genre of movies or series will be recommended new movies and series based on these features in their watch history.
These two approaches can be combined in various ways, which are called hybrid recommender systems.
Besides item metadata, user interactions can be utilized to build recommender systems. These data sources can be implicit, such as clicks and purchases or explicit such as ratings. Their quality can vary greatly while heavily influencing the performance of the system. The most reliable data is purchase data, because it can only be manipulated at a high price by actually purchasing the product. Click data and rating data can be easily distorted by bots and is thus less reliable. If carefully measured, a system can also utilize detailed consumption data for movie or music recommendations.
Common problems in recommender systems are the cold start problem and fairness. The cold start problem deals with the situation of generating recommendations for customers having little to no data about them. As recommender systems usually rely on purchase history, an empty purchase history may result in less accurate recommendations. In regard to fairness, recommender systems may also tend to produce significantly better recommendations for certain user groups. It may also happen that there is an almost exclusively recommended subset of products, for example, those sold the most.

1.2 Fashion Recommender Systems

Fashion recommender systems came to light with the growth in the fashion retail industry. Nowadays, customers are not dependent on traditional ways of buying products, such as visiting offline stores and trying out articles in dressing rooms but have the option to use online catalogs and deliver the products to their homes. With the change in this tedious process, a vast amount of options were made available to the customer creating more confusion in deciding which article to buy. Fashion recommender systems try to present customers with the top articles that are visually appealing and match their styling preferences.
Compared to other domains, the recommender system in the fashion domain is faced with major challenges. One aspect is the sparsity of the purchase data and often insufficient details about the visual appearance of the product category. To counteract this, newer approaches capture the appearance through product images, text descriptions, or customer reviews[3]. However, this requires a large data set.
In addition to this, customers want their recommendations to be of highly personalized quality. In this case, these recommender systems need to forecast customer preferences by taking into account current fashion trends, geographical location, seasonality, and social background. Analysis of this data enables recommender systems to provide their customers with high-quality recommendations[3]. Another major milestone of fashion recommender systems is to recommend not only a single article but to provide a pair of articles that would make a great outfit.
For recommender systems, input data plays a vital role in generating recommendations, and depending on the amount of information utilized, these systems can be categorized. In a base setting, transaction data, i.e. purchase history of a customer, is used and depending on the articles already bought by this customer, recommendations are generated. In a more sophisticated setting, additional information from articles and customers are utilized in order to understand underlying decision factors[3]. This additional information about articles can contain features such as colors, textures, styles, descriptions, images, etc. Information extraction on customer lifestyle can be analyzed from social media networks, fashion magazines, and websites with information on the customer and their preferences.

This report discusses the project *Graph Learning Based Fashion Recommendation* as a part of TUM Data Innovation Lab in collaboration with inovex GmbH. We worked with a public data set of H&M customers that provides detailed customer, article, and transaction data from 2018 to 2020. In order to leverage graph representation, we first perform comprehensive data exploration and then, create an expressive graph structure in a Neo4j graph database. We implemented a Random Walk approach, a Graph Embedding approach (FastRP & KNN), and a Graph Neural Network approach (GNN). Additionally, we created a web application[1] which provides the option to select among these three strategies to obtain recommendations for a specific customer. We studied the performance and explored both user-sided fairness, meaning how sensitive attributes influence the performance and item-sided fairness, meaning a sufficient variety of items being recommended. Based on our results, we discussed the potential of our approaches and graph learning in the domain of fashion recommendations.

[1] http://138.246.237.113/

2 Graph Development and Learning Algorithms

In this chapter, we explain crucial steps in order to develop a graph-based recommender system. One important component of such a system is the graph architecture. To establish a meaningful structure, we first perform an extensive data analysis. We then discuss our choice for the three learning approaches, Random Walk, FastRP & KNN, and GNN, and their theoretical foundations. In the last step, we reflect on how to evaluate the performance of our selected recommender systems.

2.1 Statistical Data Exploration

For this project, we worked on a public data set that was provided by H&M Group in a Kaggle competition [4]. The data set provides data for customers, articles, and transactions.
Transactions are provided for the time span between 20th September 2018 and 22nd September 2020 amounting to a total of 31,788,324 transactions. There are 1,371,980 customers and 105,542 articles in the data set, resulting in a sparsity degree of 0.022%.

$$\text{sparsity} = \frac{\text{total\_transactions}}{\text{total\_customers} \cdot \text{total\_articles}} = \frac{31,788,324}{1,371,98 \cdot 105,542} = 0.022\%$$

This sparsity poses a challenge when applying traditional recommender approaches like matrix factorization or collaborative filtering.
The volume of transactions usually follows a weekly pattern, where offline and online sales somewhat balance each other out. There is a more notable difference between different months as well as some spikes for special sale events such as Black Friday. After the Covid-19 pandemic started in early 2020, there is a phase of almost zero offline sales that can be explained by lockdowns and forced store closures. Sales through the online channel increased during that time. A detailed distribution of sales per channel can be seen in Figure 1.

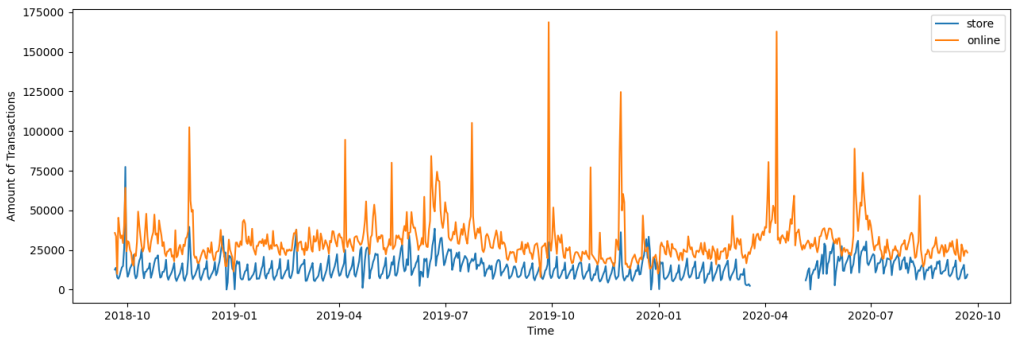


Figure 1: Amount of transactions per channel over time

Approximately 0.7% (9699) of the 1,371,980 customers have no transactions whereas around half of all customers have less than 10 transactions. All customers are either active club members (92.75%) or currently in the process of registering for the club. This suggests that the data was collected among H&M club members. Most customers are

name	values
t_dat	date of the transaction (YYYY-MM-DD)
customer_id	customer that made the transaction
article_id	article that was bought
price	scaled price (privacy)
sales_channel_id	store (1) or online (2)

Table 1: Transaction data in Kaggle dataset

between 20 and 30 years old (37%). The postal codes are very diverse even though one postal code accounts for up to 8.77% of all customers. The two next most common postal codes belong to only 0.02% and 0.01% of customers. It could be, that the one large bulk group of customers with the same postal code got assigned a default value. It is not possible to trace back the postal codes as they are only provided as hashed values.

name	values
customer_id	a unique id
Active	whether user is active for communication (boolean)
FN	whether user is subscribed to fashion news (boolean)
club_member_status	whether user is or was club member
news_frequency	how often user receives FN (NONE, REGULARY, MONTHLY)
postal_code	hashed postal code
age	integer value

Table 2: Customer data in Kaggle dataset

The article data set includes a lot of information on the category of a product as well as their graphical appearance (see Table 3). There is no information on the price as it varies over time and is thus included in the transaction data set.
The relative index distribution varies between the articles and the actual transactions suggesting that there are indices that are sold in relatively higher quantities than what we would expect in a proportional distribution. As shown in Figure 2, transactions in *Ladieswear* and *Lingeries/Tights* are over-represented in comparison to their share in all articles.

2.2 Graph Design

Modeling data through graphs helps to discover interesting relationships or patterns that might require a lot of effort to make them evident in the traditional data modeling approaches followed by tabular or relational databases. Moreover, reasoning about new or unseen data can be done more easily by leveraging limited information to connect new nodes into a larger graph. These factors made graphs very suitable to model data in recommender systems and, hence, recommender algorithms that leverage this structure offer a lot of potentials.

Figure 2: Index distribution in articles and transactions

name	amount	example
article_id	unique id	
index group	5	Ladieswear, Baby/Children, Menswear, ...
index	10	Ladieswear, Lingeries/Tights, BabySizes50-98, ...
department	250	Jersey Basic, Clean Lingerie, Tights basic,...
garment group	21	Jersey Basic, Under-, Nightwear, ...
section	56	Womens Everyday Basics, Womens Lingerie,...
product group	19	Garment Upper body, Underwear, ...
product types	131	Vest top, Bra, Underwear Tights, Socks, ...
graphical appearance	30	Solid, Stripe, All over pattern, Melange, ...
colour group	50	Black, White, Off White, Light Beige, Beige, ...
perceived colour value	8	Dark, Light, Dusty Light, Medium Dusty, ...
perceived colour master	20	Black, White, Beige, Grey, Blue, Pink, ...

Table 3: Articles data in Kaggle data set. Identification, categorization and appearance information. Example values are provided to give a better understanding of the different information.

The graph for our recommender system is a heterogeneous graph. This means that there is more than one type of node and edge, in contrast to homogeneous graphs. The two types that we will always find are customer and article. As the data set only contains purchase data and no additional customer ratings or any other implicit or explicit general interaction data, purchases are the only information that can be utilized to construct edges between the two node types identified before (customer/user and article/item). Data sets with only one type of interaction are called single-type interaction data sets. For the graph design, we assumed that items bought on the same day by the same customer belong to one basket. This basket was introduced as an additional node. This sub-graph made of the customers/users, the baskets, and the articles/items is called user-item-interaction graph [17]. Both customers and articles as well as purchases come with additional information that can be represented as node features. In our case, the additional basket node holds three features: date, total price, and the number of articles within that basket.

Article nodes hold an article id, a more detailed description, an image URL, the product name, and a product code. Customer nodes hold a customer id, the customer's age as well as information on whether the user is active and subscribed to a fashion newsletter. We added additional nodes for metadata of articles and customers if the value of that data is not unique. The data set provides further information for each article that is used to build a domain knowledge graph. An article node has relationships to nodes of type graphic appearance, color (master color, color group, color value), and garment group. In addition, we constructed a hierarchy for nodes of type product type and product group, where only product type is connected to nodes of type article. A second hierarchy was built for the indices and index groups. A detailed explanation for the possible values of the different node types can be found in subsection 2.1.
Some of the additional knowledge about customer nodes is represented in another sub-graph called social-relation-graph. Age group nodes were created grouping users by their age in 5-year steps from 15 to 100. In addition, all customer nodes are connected to a postal code node, adding further context on where a customer is from. Customer nodes are connected to similar customers in the social-relation-graph via age groups and postal codes.
Finally, basket nodes are connected to some ontology knowledge nodes, namely the sales channel and the season in which the transaction occurred.

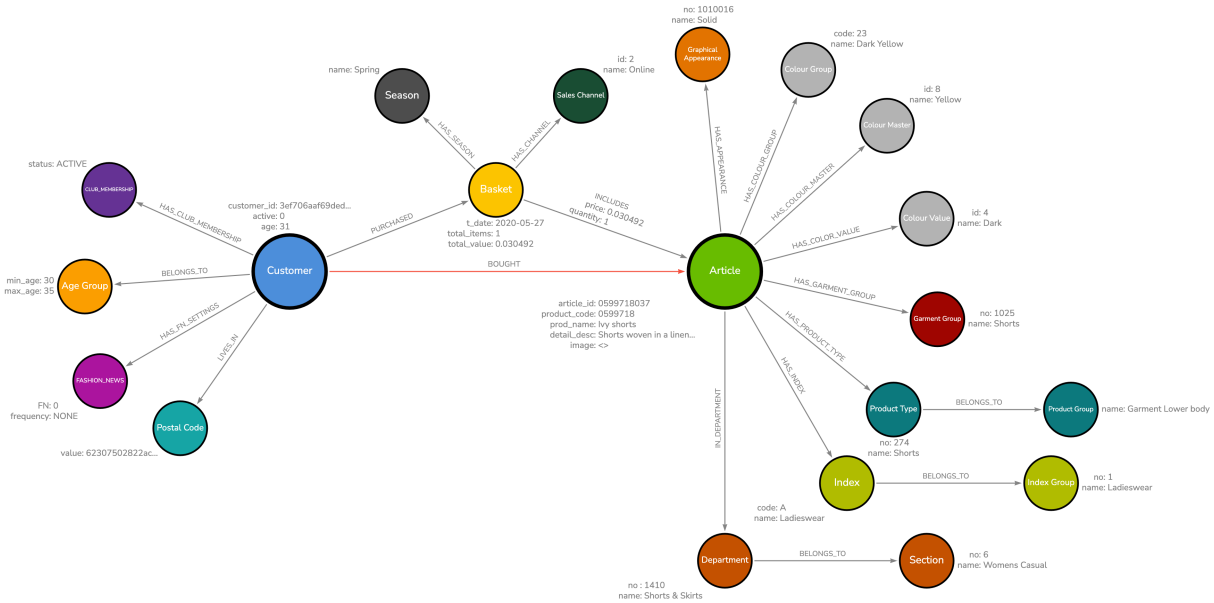


Figure 3: Full graph used by the recommender system

Figure 3 shows the graph in full detail with all its different nodes and relationships. All the previously described sub-graphs are part of this graph. We decided to represent all non-unique features as nodes and relationships in the graph to leverage the full potential of graph-based learning.

2.3 Graph Learning Approaches

Most machine learning approaches deal with euclidean data like images, sound, or text. However, many real-life problems can be abstracted into a graph to represent the information in nodes with features while also introducing edges to represent relationships in the data.
That allows for a variety of new approaches, that purely leverage topology (graph structure) as well as machine learning models that optimize for a certain function.
As graphs are homomorphic, meaning that we can have a mapping between the nodes of one graph to the nodes of another graph such that the edges are preserved, there is an inductive bias by representation if we were to pass the graph into a machine learning model that deals with euclidean data.
Graphs can be useful in a variety of different domains. Graphs representing molecular structures can use graph classification algorithms to predict the properties of a molecule. Node classification algorithms can be used to identify fraudulent nodes in networks. Community detection in graphs can be used to identify clusters in a graph. Social networks can use these communities to identify users that share common interests.
In this project, we worked on link prediction, which aims to predict relationships of type *BOUGHT* between articles and customers.
Graph learning approaches can be divided into three types [17]:

- **Random Walk** approaches use a random walker to traverse the graph starting from a defined start node. The nodes visited on random paths contain information about the surroundings of the start node.
- **Graph Embedding** approaches use a trainable embedding function to embed nodes into an n-dimensional vector space using their own features and all the connected edges and nodes. The embeddings can be used in euclidean inference approaches.
- **Graph Neural Networks** use message passing to aggregate information from surrounding nodes based on aggregation functions with learnable parameters.

2.3.1 Random Walk

The most popular application of a random walk approach is the Google Page Rank algorithm, which is based on a random walker, respectively a random surfer randomly surfing webpages to get the overall importance of a webpage. The random surfer can also be modeled using a probabilistic transition matrix. Multiplying the matrix by itself models a single step. The eigenvector of the transition matrix corresponds to the stationary distribution of the Markov Chain that is represented by the matrix [14].
If we limit the number of steps our random walker takes through the graph, we can gather information about the neighborhood of the starting node rather than the whole graph. In our recommender system, we start the random walk from a customer node for which we want to generate recommendations. After a finite number of steps in the graph described in subsection 2.2 we get a set of visited nodes. Not all these nodes are of type article and we will also encounter a higher number of articles that are connected to the customer through a direct relationship (i.e. those bought in the past). Therefore, the visited nodes

are filtered for article nodes and articles that did not appear in the customer's purchase history. The top 10 most visited articles are recommended and ranked by their visits.
Our Random Walk uses a graph projection (in-memory representation of the graph used by the Neo4j GDS library[2]) which does not consider the direction of the edges.
The Random Walk is executed using Neo4j's Graph Data Science library that comes with an out-of-the-box Random Walk implementation[3]. Executing the Random Walk in the database improves performance significantly, as the communication between the database and the recommender system is very time-consuming.

2.3.2 Graph Embedding

Graph Embedding approaches leverage the graph structure or topology to convert nodes into low-dimensional vectors. These vectors, also known as embeddings, reflect the graph structure, meaning that two nodes with similar neighborhoods will be assigned similar embeddings. To generate recommendations based on this strategy, we decided to obtain the most similar article embeddings for a specific customer. For this approach, we have leveraged functions provided by the Neo4j Graph Data Science Library. As a first step, we create graph projection or in-memory projection in Neo4j taking nodes (customer, article, postal_code, age, news_frequency, club_member_status) along with all the relationships that exist between these nodes.
For the embedding generation, we decided to use the Fast Random Projection (FastRP) algorithm. The major reason for this choice is that FastRP has a good performance by using sparse random projections [2]. Along with this, it is easily scalable since embedding computation is performed iteratively and it handles limited memory well [4]. The node embeddings generated by this algorithm capture the topological information of the graph. This implies that two nodes with similar neighborhoods will be assigned similar embedding vectors. These node embeddings are dependent on the radius of the neighborhood which is given by the number of iterations performed by the algorithm [5]. For example, with the first iteration, only the direct neighbors of the node are considered for the node embedding and as we proceed with the iterations, neighbors of higher degree are considered which gives the embedding at the later stage.
In the first attempt, we used the KNN functionality by Neo4j to identify customers that are similar to a given customer with the help of the FastRP embeddings generated in the previous step. After getting the customer with the highest similarity score, we recommend articles from the purchase history of the similar customer [6]. However, we realized that this approach does not work if the customer with the highest similarity score is a cold start customer. In this case, the customer has no purchase history that we could recommend for the given customer. This limitation arises even more likely if the given customer is a cold-start user himself.
To address the problem regarding cold start customers, we decided to use filtered KNN by Neo4j and modified our approach to obtain recommendations as per similarity between

[2]https://neo4j.com/docs/graph-data-science/current/management-ops/projections/graph-project/
[3]https://neo4j.com/docs/graph-data-science/current/algorithms/random-walk/
[4]https://towardsdatascience.com/getting-started-with-graph-embeddings-2f06030e97ae
[5]https://neo4j.com/docs/graph-data-science/current/machine-learning/node-embeddings/fastrp/#algorithms-embeddings-fastrp-syntax
[6]https://neo4j.com/docs/graph-data-science/current/end-to-end-examples/fastrp-knn-example/

a customer and an article instead of within customers. Filtered KNN extends the classic KNN allowing us to calculate similarities between specific source and target nodes. For every customer, we use filtered KNN to identify articles whose FastRP embeddings are high in similarity score to the customer embedding and then provide a list of recommendations with these articles.

2.3.3 Graph Neural Network

Graph Neural Networks (GNN) are a machine learning technique that optimizes for a task (e.g. classification, regression) leveraging data graph structure. They create vector representations or embeddings for all nodes in a graph, using operations known as graph convolutions, so that neighborhood-aware embeddings, which capture knowledge from nodes and relationships, are created. These embeddings can be used to perform different tasks such as node classification, graph classification or, as in our case, link prediction. We decided to model the recommendation problem as a link prediction task where we try to predict the probability that an edge between a customer and an article exists, in other words, the probability that the customer will buy that article.

Model Architecture

The common architecture to perform link prediction between two nodes in a graph is using their embeddings as an input to a regular neural network that outputs a probability (i.e. edge probability) that the two nodes are connected [16].

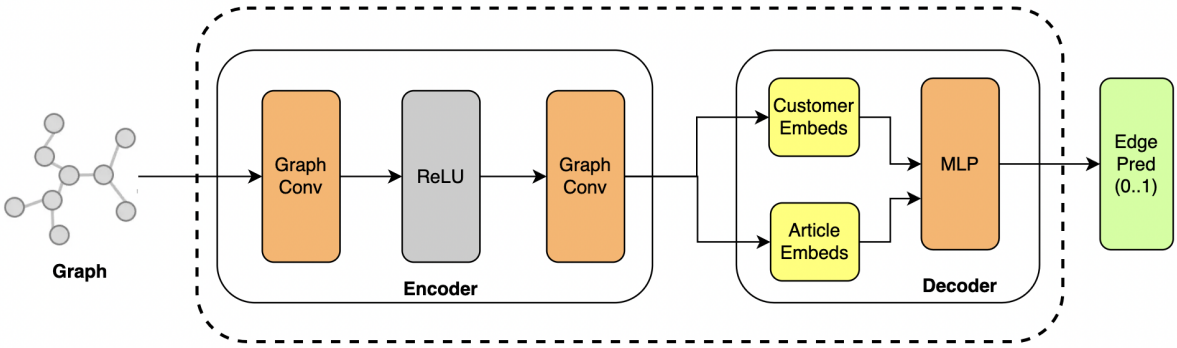


Figure 4: GNN Architecture for Customer-Article Link Prediction

There are multiple components that can be configured in this architecture, but the one that can affect the performance the most is the type of graph convolution selected to create neighborhood-aware embeddings.

Graph convolutions implement an iterative process called *message passing* so that for each node: 1) information is collected from neighbors, 2) this information is aggregated, and finally, 3) the node embedding is updated. One type of graph convolution is GraphSAGE, which is characterized by two factors [8]: neighborhood sampling and learnable aggregation functions. Neighborhood sampling refers to the fact that GraphSAGE convolutions do not use all the node's neighborhood to compute embeddings, but rather a sample of it, making the overall message passing process faster and scalable when dealing with large-scale graphs [19]. On the other hand, using learnable aggregation functions, instead of

regular operations (e.g., mean or sum), improves the generalization power. Considering these factors and the scale of our graph, we considered GraphSAGE the most suitable choice when implementing our GNN approach.

Training approach

To train a model, one always needs to split the data into training, validation, and test sets. Splitting our data for link prediction is not trivial, since we are dealing with a transductive learning problem. Inductive learning is what is more commonly known in machine learning tasks, where a labeled data set is given that can be split into distinct sets for training, validation, and testing with the goal of generalization to unseen data. In graph learning, this would mean having a set of graphs and performing graph-level prediction. In this case, the data set can be split as usual. In transductive learning, both labeled and unlabeled instances are given and the goal is to primarily predict the labels on the unlabeled instances. The challenge is that the unlabeled instances are - unlike in inductive learning - already observed during training [10]. This is exactly our case since we have only one graph and want to predict edges between nodes, which we already use during training. The graph can be observed in all data set splits.

One recommended approach is to split all edges between customers and articles into 4 disjoint sets called training message edges, training supervision edges, validation edges, and test edges. At training time, the training message edges are used to predict the training supervision edges. At validation time both the training message edges and training supervision edges are used to predict the validation edges and finally at test time, all previous edges are used to predict the test edges. This ensures that the test edges are not included in the training and validation steps and also that the validation edges are not included in the training step [10]. Following this approach, we decided to split the *BOUGHT*-edges into 80% for training, 10% for validation, and 10% for testing. We used 70% of the training edges for message passing and 30% for supervision.

Given that our task is to predict whether or not an edge exists (i.e. *BOUGHT* edge) between a customer and an article with a certain probability, we decided to use binary cross-entropy, which is a function used for binary tasks. To make use of this function we need a ground truth set with positive and negative edges. However, the way the graph is built, we only have positive edges with existing customer-article relationships (i.e. *BOUGHT* edges), indicating articles that the customer has bought in the past. Therefore, we need to obtain negative edges so that the GNN can learn to accurately predict positive ones.

The process of generating negative edges is called negative sampling and there are multiple strategies with different degrees of impact [9], but the most simple one is negative random sampling, where negative edges are randomly sampled at each training batch. Generating different negative edges at each training batch avoids negative bias towards certain target nodes and, thus, focuses on learning to predict positive edges accurately. Due to its ease of implementation, we decided to use negative random sampling.

To evaluate the training performance and determine whether or not we have obtained a satisfactory model, we use the ROC curve (Receiver Operating Characteristic curve) and its corresponding AUC (Area under the ROC Curve). Both are used to evaluate the performance of classifiers and they measure how well a model can distinguish between the true positives and negatives. The ROC curve plots the true positive rate (TPR) against

the false positive rate (FPR) at different classification thresholds. The further the curve is from the diagonal line, the better the model is at discriminating between positives and negatives.

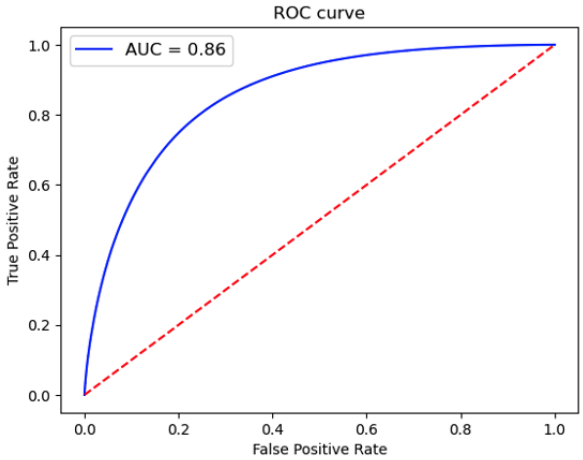


Figure 5: GNN Model ROC Curve

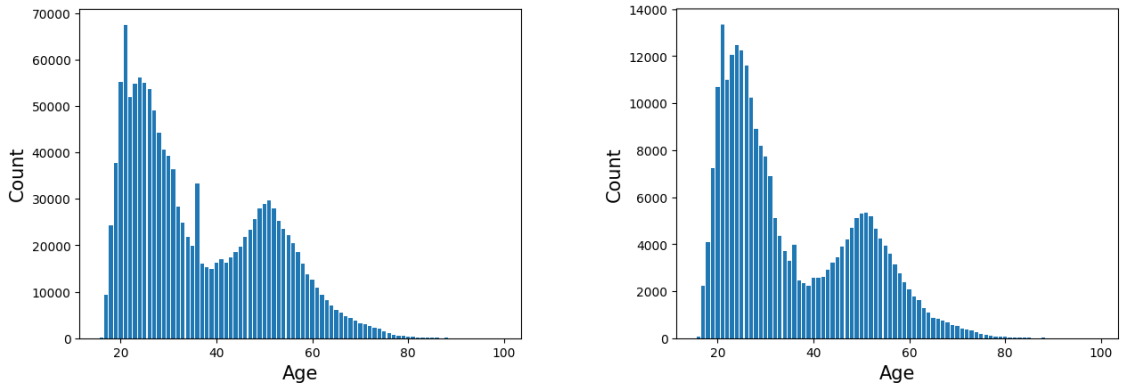
To further summarize the whole ROC curve, the AUC metric can also be calculated. The more AUC is close to one, the better the classifier. Our model reached an AUC of 0.86. After obtaining a trained model, the end result is that for each customer we can get purchase probabilities against any article in the catalog, where the recommendation will be the highest probability.
Taking a deeper look into the probabilities distribution, it is noted that it is highly dependent on the length of the purchase history. Customers with a low amount of bought articles are predicted a very few amount of articles with high probability, in contrast to customers with a large amount of articles in their purchase history.

2.4 Evaluation Strategy

Evaluating the performance of a recommender system is a challenging task because multiple stakeholders with sometimes opposing demands, like customer and provider, need to be considered. For example, some customers prefer that the articles recommended to them match their long-term preferences, while others prefer complementary or alternative items. Looking into the provider perspective, the system should, among other criteria, increase user engagement, business success, or the number of items per basket. These diverse interests need to be reflected in the evaluation of the recommendation algorithms. Furthermore, we need to split our data into training and test sets in a meaningful manner. Therefore, we first discuss our choice for the data split and then explain how we will compare the quality. Additionally, we investigate the fairness of the system in terms of user and item attributes, including the group of cold start customers.

2.4.1 Performance Metrics

An important part of the evaluation strategy is the split of our data into training and test set. This is not trivial for recommender systems and splitting techniques focus on



(a) whole data set (b) test set

Figure 6: Customers' Age Distribution

different aspects of the data. Common approaches are Leave One Last, which removes the final transaction per customer, Temporal Split, which performs the split based on historical data, User split, which removes specific users and all their transactions from the training set, and Random Split[13].
We decided to follow a Temporal Global split strategy since this was also used in the Kaggle competition for evaluation [4]. Consequently, we removed the last month of transactions (i.e. from 22 August 2020 until 22 September 2020) from the data set for evaluation and the remaining periods (i.e. from 20 September 2018 until 21 August 2020) were used in the setup of the three graph approaches. Thus, we ensure all approaches are evaluated on unseen data.
Employing a temporal split, we can only use those customers for the evaluation, which bought at least one article within that last month. We chose a time span of one month for two reasons. Firstly, it is desirable that recommended articles should be predicted for a short period of time and get updated with the season, new fashion trends, and other time-dependent factors. Therefore, a short evaluation period makes sense. Secondly, the test set of customers should be able to represent the whole data set. By choosing a time span of one month, we can still evaluate the performance for 256,355 customers which make up 18% of the total customers. Figure 6 also shows that the age distribution within the test set is comparable to the overall age distribution. This is important because later we want to evaluate the performance across the different age groups.

Mean Average Precision at K
Our fashion recommendation system generates a list of recommendations for each customer. To ensure that customer attention is captured at the beginning of the recommendation list, an ordered list of articles is provided. Mean Average Precision at K (MAP@K) is used to understand how good the recommendations generated by our system are, along with the order in which the articles are recommended.
In the context of recommendation systems, precision gives the measure of relevant recommended items out of the items recommended by the system. We take precision at K to consider only the subset of recommendations i.e. top K recommendations from the list. For all the customers in the test set, we calculate the average precision at K (AP@K)

using the precision of articles in the recommendation list. In order to get the MAP@K, we take the mean of AP@K for all the customers in the test set [12].

$$\text{Precision@K} = \frac{\text{Number of article recommendations that are relevant in top K}}{\text{Number of articles that are recommended}}$$

$$\text{AP@K} = \frac{1}{M} \sum_{k=1}^{K} \left(\text{Precision@}k \text{ if } k^{th} \text{ article was relevant}\right),$$

where M is the number of article recommendations that are relevant for the customer, i. e. the customer has bought these articles in the test month.

$$\text{MAP@K} = \frac{1}{N} \sum_{n=1}^{N} (\text{AP@K of Customer(n)}),$$

where N is the number of customers in the test set

Catalog Coverage
The quality of the recommendations should not only be measured by the predictive accuracy. Instead, research shows that other metrics like coverage is more proficient in reflecting the perceived performance of the recommendations by customers[5]. If a system covers wide parts of the catalog and thus achieves high catalog coverage, users are exposed to an extensive range of fashion articles and are therefore able to discover new items more easily. Providers of fashion items target a higher coverage, because it results in both an improved shopping experience and higher engagement of the customers and thus more sales[11]. Therefore, we decided to observe the coverage, which is calculated by dividing the number of distinct articles in the recommendations across all users by the number of distinct articles in the catalog.

2.4.2 Fairness Metrics

Besides these metrics, fairness is one important part of our evaluation strategy. Fairness has become more important for machine learning tasks, including recommender systems. Special care must be taken by fashion recommender systems to not reinforce social biases and perpetuate long-observed behaviours[3]. There exists no clear definition of fairness, however, fairness requirements can be sectioned into group and individual fairness. While group fairness states that protected groups defined by a sensitive attribute should be treated similarly as advantaged groups, individual fairness aims for a similar treatment of similar individuals [18].

Customer Fairness
Given our specific recommendation task and data set, we focus on group fairness on both the customer side and the article side. We split our test-user set into several groups according to their age and then compare the quality of the recommendations given to each group. Enforcing age fairness means that systems do not discriminate against any age group and treat all groups equally by generating useful and personalized recommendations. This is important because age can be a sensitive attribute, and at the same time

contains valuable information in order to provide relevant recommendations. Observing and understanding age fairness helps in the process of building both trustworthy and effective recommenders. Consequently, we calculate the MAP@K for each age group. Afterwards, we compare these results for our recommender systems and identify any notions of unfairness. In a further step, one could apply a re-ranking in order to improve fairness among the age groups[11]. However, this lies beyond the scope of the project.

Cold Start
The cold start problem can also be considered a customer fairness related issue. Here, we group our customers according to the length of their purchase history into new customers and already-known customers. We have 256,355 customers in our test set and about 10% of these have made no purchases within the training time span (20 September 2018 to 21 August 2020). For this group of new users, we compare the quality of the recommendations made using MAP@10. This should give an idea of how well our recommender strategies deal with the cold start problem.

Item Fairness
Besides customer fairness, we evaluate the article exposure. It is a well-known problem, that already popular items get recommended at a higher rank, and due to that higher exposure become again more popular[18]. It remains open whether this is problematic or desirable for customers since popularity might be caused by trends, pricing, and other factors. In any way, this popularity bias issue highlights how much influence the exposure has on the recommendations in the long term. This bias highly correlates with the catalog coverage, but we furthermore want to examine how the exposure behaves across article groups. For this purpose, we split the recommended items according to their index and count the occurrence of articles per index. To compute the percentage exposure, we then divide this number by the count of articles per index within the catalog[11].

$$\text{exposure(index)} = \frac{\text{count(index)\_recommendations}}{\text{count(index)\_catalog}}$$

In Figure 2, we give an overview of the indices of the catalog and their occurrence in both the catalog and transactions. We chose to separate the articles according to the index attribute because it correlates with the customer groups - Children, Men, Women, and General - which means, we not only gain more insights into group fairness but also get insights into the diversity of the recommendations.

3 Technical Setup and Application

Throughout the project, we have used different frameworks and applications as well as computational and data resources. In the following, we will give an overview of our technical setup in regard to data storage and retrieval, the GNN model training, the interactive application, and finally, we will explain how the different parts are connected in our recommender system architecture.

3.1 Database and Graph Data Science

For our data storage and retrieval purposes we decided to use a graph database. A graph database is a type of NoSQL database that stores data in the form of nodes and edges (relationships) in contrast to a relational database where data is stored in tabular format. Graph databases are well suited for handling real-time data where one can add new nodes and relationships between nodes. These databases can traverse data quickly without performance damage[7].
In our project, the nature of our data and the relationships that exist between different entities made the graph database a logical choice. With the help of a graph database, we can easily access the articles bought by a customer or get the customers who have made a transaction in a certain period.
We have thoroughly evaluated several popular graph databases such as Nebula Graph, Neo4j, OrientDB, and ArangoDB and have decided that Neo4j is the best fit for our requirements.
The major reason supporting our choice of database is that Neo4j is an open-source database that provides us with vast resources in terms of documentation and it comes with a Graph Data Science library (GDS) with built-in functionalities to analyze graph structures[8]. We have utilized the following functionalities for our Random Walk approach and Graph Embedding (FastRP & KNN) approach:

- Path Finding: These algorithms help find the shortest path or evaluate the availability and quality of routes. For example, randomly traverse the graph to obtain articles that a customer is likely to buy.
- Node Embeddings: For nodes in our graph database, we require these algorithms to compute low dimensional vector representations of nodes in the graph. These vectors are called embeddings.
- Similarity: These algorithms help calculate the similarity of nodes based on the neighborhood or the node properties. Neo4j provides several similarity functions such as euclidean distance or cosine similarity.

3.2 Graph Training Set-Up

Two of our three graph approaches - random walk and graph embedding - use graph algorithms that do not need training. Conversely, for the GNN approach, we need to

[7] https://www.techtarget.com/searchdatamanagement/feature/Graph-database-vs-relational-database-Key-differences
[8] https://neo4j.com/docs/graph-data-science/current/algorithms/

optimize a model based on a loss function and in the following section, we explain how we dealt with training such a model using a large-scale graph.
For building the model we decided to use PyTorch Geometric[9], which is a library based on PyTorch specifically for graph structures. It supports building both homogeneous and heterogeneous graphs, and it has built-in GNN operators such as GraphSAGE. Furthermore, it offers functionalities for mini-batching and random sampling for link prediction, which also supports negative sampling.
All model hyperparameters (e.g. batch size, learning rate) are adjustable through a hydra configuration file. Then, the graph in PyTorch Geometric is built directly using the CSV files provided by H&M. The GNN approach does not need to communicate to the database during training, since the Neo4j functionalities are not needed, and everything is done through PyTorch Geometric.
To train our GNN approach we used the computational resources by the Leibniz Supercomputing Centre (LRZ). This was necessary since training with 100% of data (approx. 11M nodes and 80M edges) exceeded the computational power of our local machines. To prepare for training, we used an NGC docker container from NVIDIA and customized it with our specific package versions. Furthermore, we used SLURM to submit our SBATCH jobs on LRZ and trained on a DGX A100 GPU from NVIDIA with 80GB Memory.
After training, the model binaries can be used to obtain recommendations in our main recommender architecture.

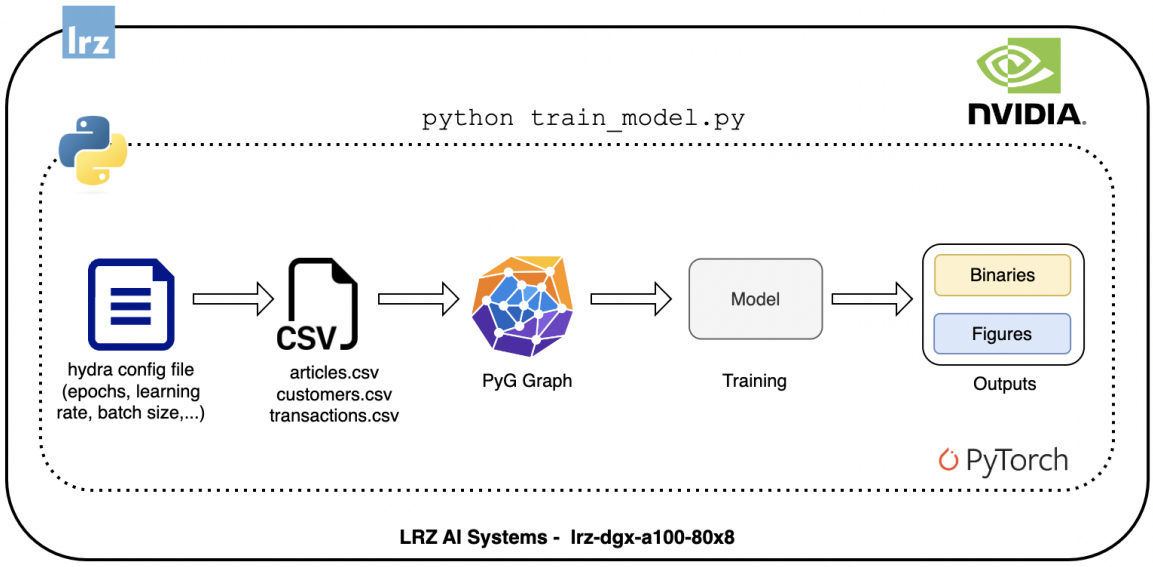
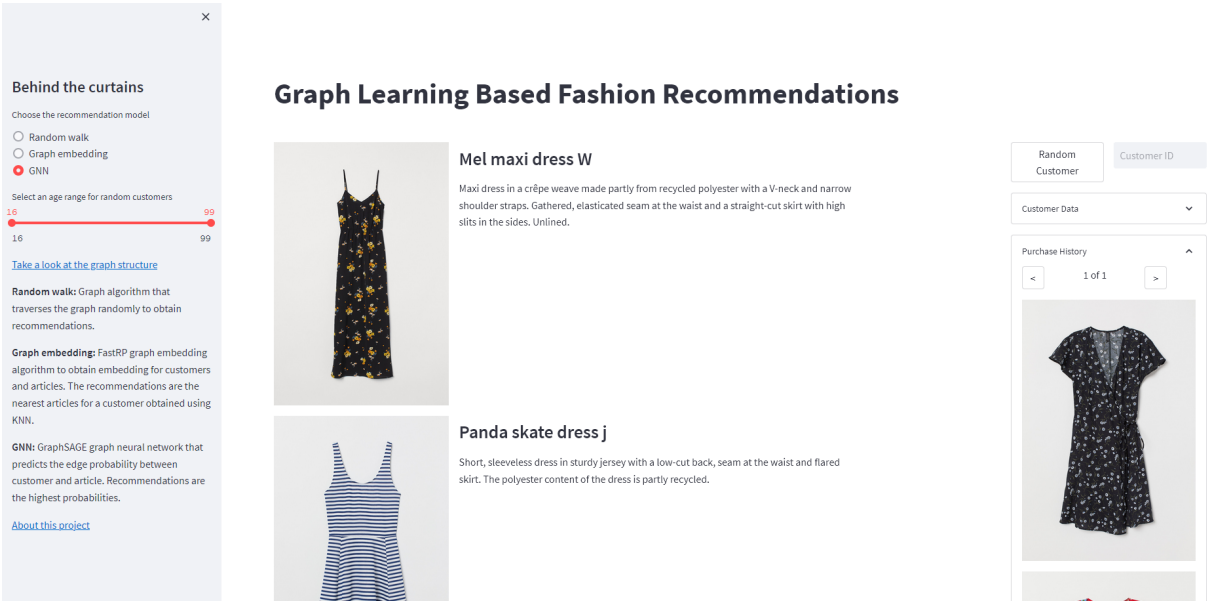


Figure 7: GNN Training

[9]https://pytorch-geometric.readthedocs.io

3.3 Graphical User Interface

In order to be able to visually compare the recommendations of our different graph approaches, we wanted to build an interactive GUI. We decided to use Streamlit[10] - an open-source framework to create web applications in python.
Streamlit is specifically made for machine learning and data science purposes and works in a straightforward way. Whenever a user interacts with the application (e.g. button click) the entire script is rerun from top to bottom. Furthermore, it is possible to define customized callback functions and safe values throughout different session states.
Our GUI is split into three areas. On the right side is the customer area. The user can search for a specific customer by entering their customer id or by requesting a random customer instead. The customer data (e.g. age, postal code, ...) and the purchase history of the corresponding customer is shown.
On the left side is the "Behind the curtains" area, with an overview of the graph architecture and the different graph approaches. The user can select the preferred recommendation strategy, specify an age range for the next random customer and receive more information about this project. Finally, in the center, 10 recommendations are shown for the specific customer and the chosen graph approach.



3.4 System Architecture

The recommendations are obtained within our Streamlit web application by communicating with two Neo4j databases.
The application database is used to receive content related to customers (e.g. status, membership, postal code) and articles (e.g. name, description, image URL). The image URLs point to an Object Storage where the image files are stored.
The training database is used to run graph algorithms with enough resources in order to generate recommendations for the Random Walk and Graph Embedding approaches.

[10] https://streamlit.io

For GNN-based recommendations, model binaries are included in the application. These binaries contain the trained model and pre-computed node embeddings to speed up the inference time.

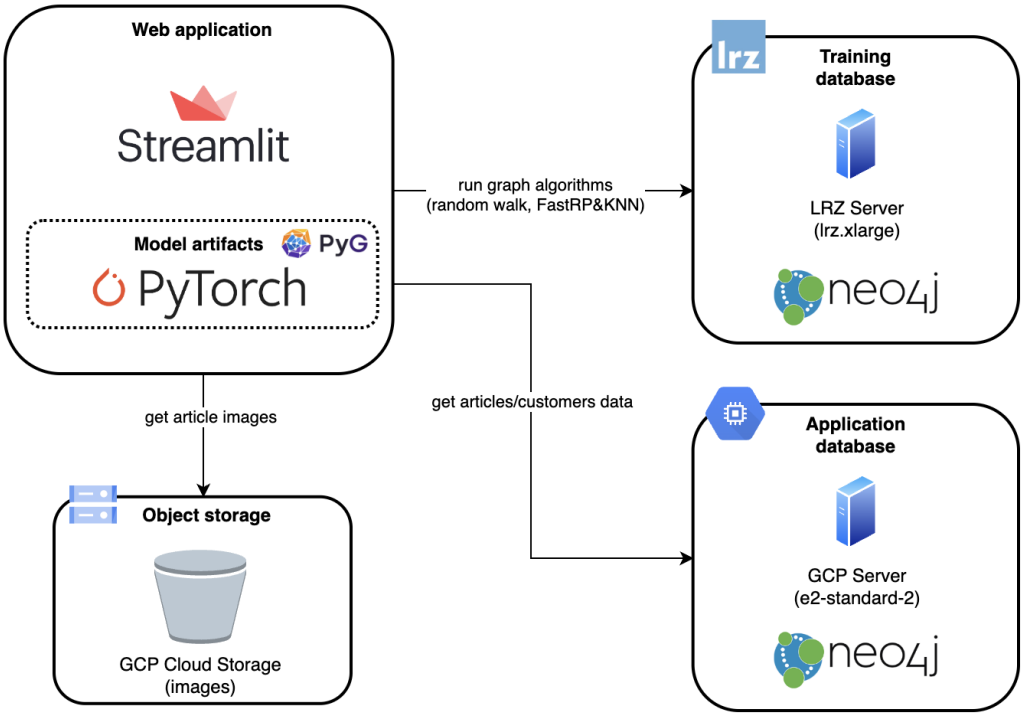


Figure 8: Recommender architecture

4 Results

Finally, we perform an analysis of our fashion recommender approaches following the evaluation strategy from subsection 2.4. First, we compare the results from the Random Walk, Graph Embedding (FastRP & KNN), and GNN approaches and discuss their relevance for the fashion industry. Then, we attempt to interpret their performance and identify both limitations and advantages of the models.

4.1 Recommendation Quality

First, we study the MAP@10 scores of our approaches as illustrated in Table 4. In general, the MAP@10 value seems low. However, bringing this score into context, a MAP value equal to 1 would mean the system recommends for each customer one correct article at the first position, i.e. an article bought within the test month. Considering a catalog size of 105,542 and a set of 256,355 test customers, this is very challenging. The highest MAP@10 value is achieved by the GNN approach improving the baseline approach by 3934%, while the MAP@10 score for FastRP & KNN is 45% lower than the Random Walk.

	Random Walk	FastRP & KNN	GNN
MAP@10	1.7359e-04	7.7552e-05	7.0197e-03
MAP@10 (Cold Start)	2.2523e-04	2.7372e-05	2.9817e-02
Catalog-Coverage	0.8545	0.8080	0.0354

Table 4: Evaluation results for the test set

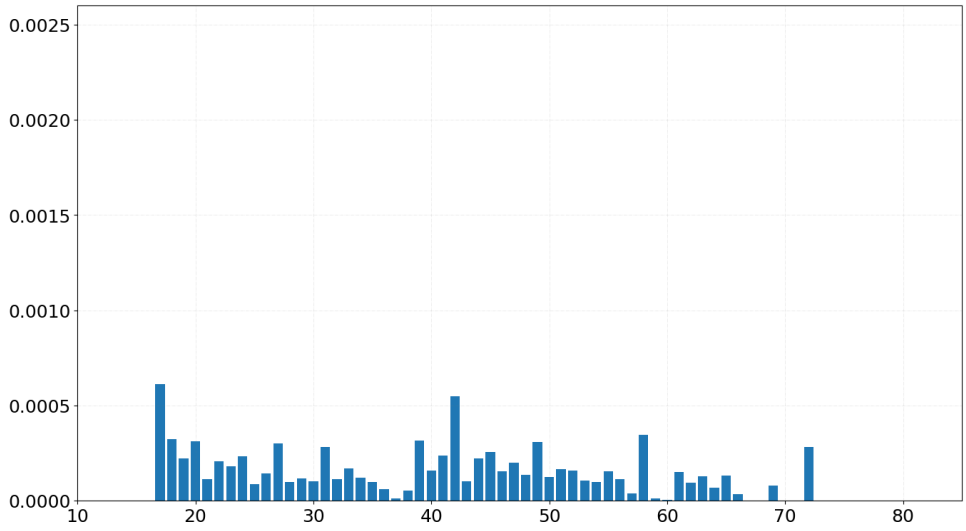
One would expect that the FastRP algorithm performs better than our baseline approach since it is more sophisticated. The poor performance of the FastRP algorithm may stem from the small embedding dimension of 32. A higher dimension could capture more topological information from the graph, leading to improved identification of new relationships between customer and their potential article recommendations. This could be a possible factor contributing to the algorithm's low MAP value. *A value of at least 256 gives good results on graphs in the order of* 10^5 *nodes, but in general increasing the dimension improves results. Increasing the embedding dimension will however increase memory requirements and runtime linearly*[11]. Due to this, our initial plan was to use at least 256 as embedding dimension. However, the server configuration of our database did not match the memory requirements of the KNN algorithm. Along with this, we wanted to compare the same embedding dimension as used in GNN and therefore, decided on a dimension of 32.
Another reason, why the MAP@10 value of both Random Walk and FastRP & KNN differ so strongly from the one for the GNN approach, is that they lack specific optimization for the *BOUGHT* relationship. While the GNN approach optimizes for a correct prediction of bought articles using a binary cross-entropy loss, neither the Random Walk nor FastRP & KNN are trained with the goal to predict future purchases and instead exploit the

[11] https://neo4j.com/docs/graph-data-science/current/machine-learning/node-embeddings/fastrp/#algorithms-embeddings-fastrp-syntax

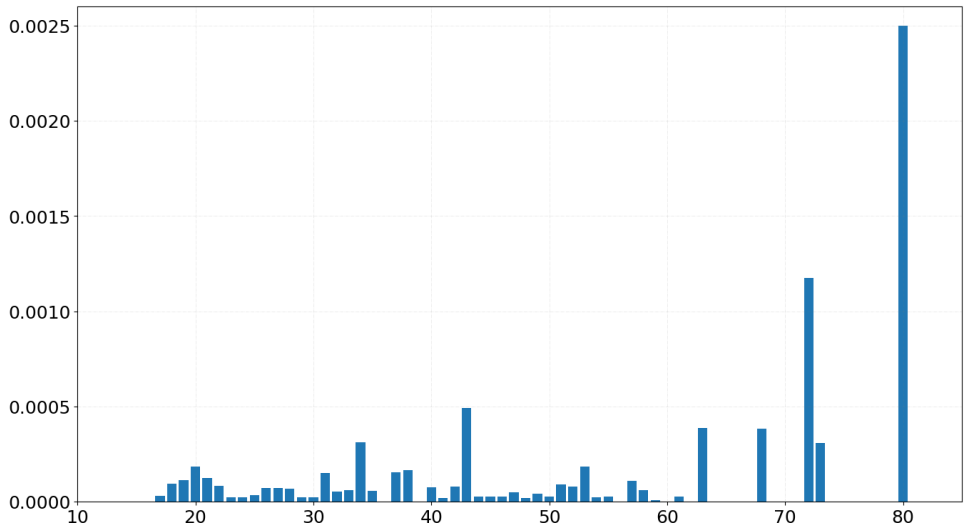
graph's topology and node similarity. However, this optimization done by the GNN does not consider any notion of ranking, making it difficult to obtain a high MAP value.
To capture further aspects of the users' shopping experience, the second element of our evaluation strategy is the catalog-coverage. As summarized in Table 4, the coverage achieved by the Random Walk and FastRP & KNN approach is similar with a value of 80-85%. Meanwhile, the recommendations provided by the Graph Neural Network approach only cover 3.5% of the catalog. This extreme difference shows that the trained GNN model recommends only a few selected items, probably because they are most likely to be bought or are already popular. Consequently, the model can realize a better MAP value but at the cost of low coverage. Especially from a business perspective, one would like to achieve a high catalog-coverage[5].

4.2 Fairness Performance

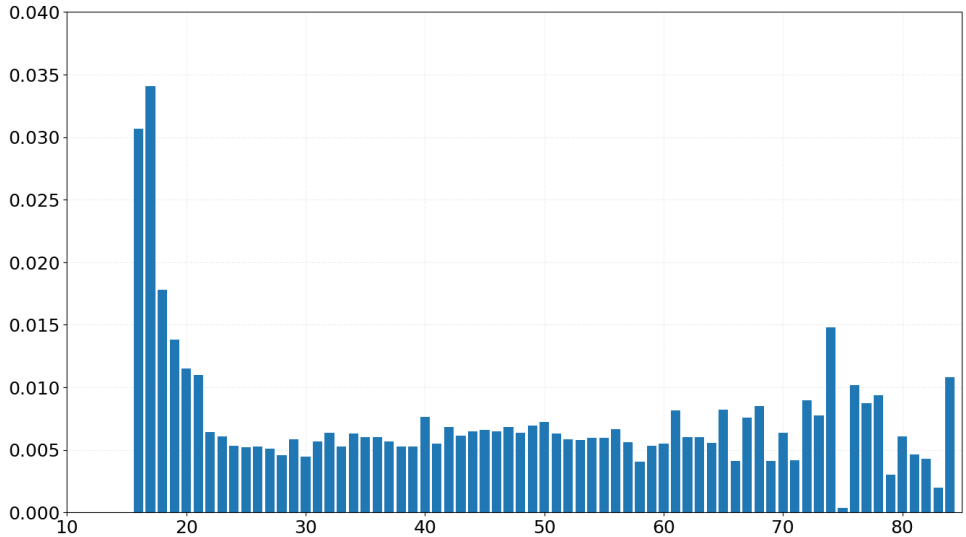
Besides MAP and coverage, our evaluation strategy focuses on fairness metrics, in particular age fairness and index exposure. We investigate whether the quality for different age groups is similar, which also helps us to understand the strengths of the recommendation strategies. Furthermore, the distribution of fashion pieces across the indexes can give us an intuition of how diverse the recommendations are.
Figure 9 shows that the MAP@10 is relatively evenly distributed among the ages for the Random Walk. We interpret this result that the Random Walk - due to its nature - does not favor any age group, and the age feature might only have a minor influence on the final recommendation. Looking at the distributions for the FastRP algorithm, we notice that the MAP@10 for specific customer ages, 71 and 80, is especially high, with a maximum of 0.0025, while in between the MAP@10 is equal to 0.
One explanation for this uneven distribution could be the number of customers per age included in both, our training set and test set. The data set consists of many customers aged between 20 and 30, while there are very few older than 70, as summarized in Figure 6. Specifically, we have 55839 customers aged between 20 and 25, but only 87 users aged between 80 and 85 in our test set. This age distribution might influence the performance and evaluation of the FastRP & KNN approach. Our theory would be that the algorithm deals better with input consisting of fewer customers with similar, and therefore predictable shopping behavior, while many diverse styles among young people lead to an information overload. To support this theory, we examine the purchase article distribution for both age groups, younger than 30 and older than 70. As highlighted in Figure 12, for the younger customers, the two most popular groups make up 65% of the total items, while for the elderly customers the two most popular groups already make up 80% of the total items.
Conversely, the GNN model performs best for young customers aged between 15 and 20. This could mean that the GNN approach can leverage all given information and their diverse shopping behavior. Besides, the MAP@10 is almost continuously and evenly high for all customer ages, which is a very promising result.
For all three approaches, we need to mention that the MAP for people aged above 85 was 0, probably due to the very few customers within the data set. Special effort would be needed to understand their shopping preferences in order to recommend personalized fashion items.



(a) Random Walk



(b) FastRP & KNN



(c) GNN

Figure 9: MAP@10 per age. *Notice the different y-scale for the GNN result.*

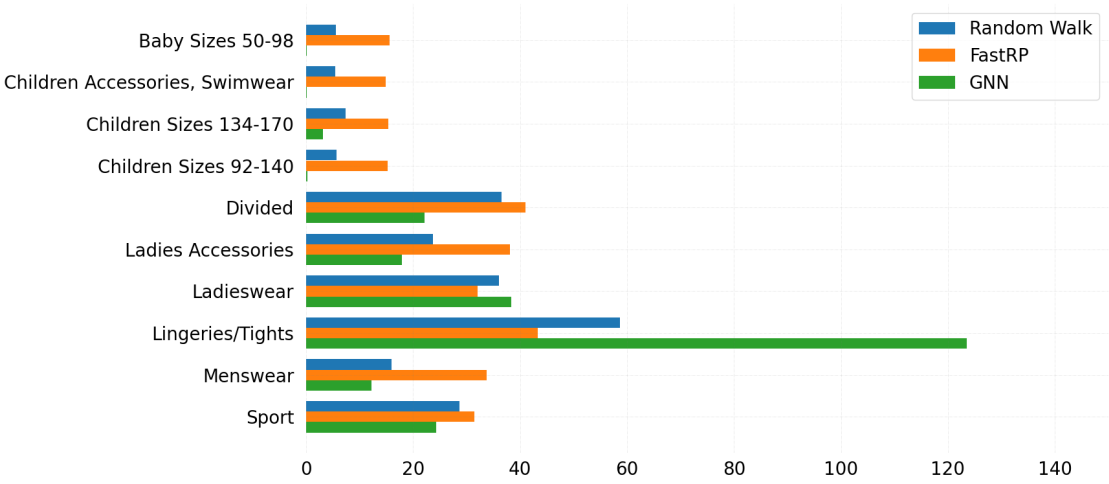


Figure 10: Article exposure per index

Next, we consider the cold start problem and assess the performance in terms of MAP@10 for new users. Interestingly, both the Random Walk and the GNN approaches achieve a higher MAP@10 value in the cold start setting, see Table 4. Based on these improvements in precision, we hypothesize that it may be easier to recommend articles not based on a long purchase history but instead recommending popular items given such a sparse data set. This theory can be supported by the training results for the GNN model.

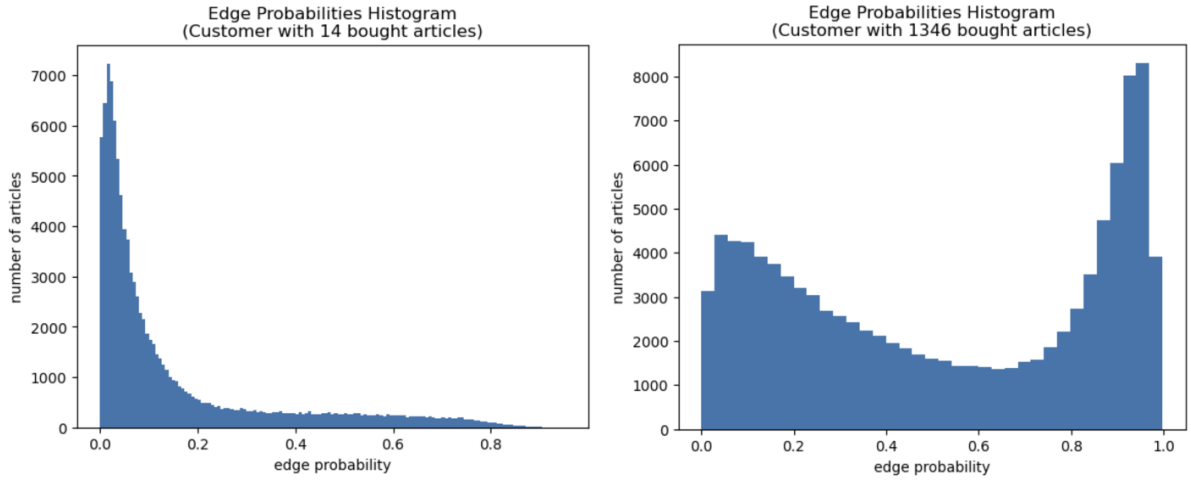


Figure 11: GNN edge probabilities

In Figure 11 one can observe the differences between the probability distributions for a customer with a small purchase history and one with a large purchase history. Hence, it seems the GNN is more certain about which articles it should predict for customers with a small purchase history.
The index distribution of purchases in the training period (see Figure 12) directly influences the index distribution of the recommended articles of our three graph strategies.

Index distribution in training period

Age: >= 70

index	total_articles	pct
Ladieswear	47280	64.09
Divided	10919	14.80
Ladies Accessories	4580	6.21
Lingeries/Tights	4492	6.09
Menswear	3708	5.03
Sport	999	1.35
Children Sizes 92-140	686	0.93
Children Sizes 134-170	527	0.71
Baby Sizes 50-98	405	0.55
Children Accessories, Swimwear	178	0.24

Age: <= 30

index	total_articles	pct
Ladieswear	2077431	39.47
Divided	1301701	24.73
Lingeries/Tights	986766	18.75
Ladies Accessories	353588	6.72
Menswear	255510	4.85
Sport	205979	3.91
Baby Sizes 50-98	32580	0.62
Children Sizes 134-170	21717	0.41
Children Sizes 92-140	21270	0.40
Children Accessories, Swimwear	6435	0.12

Figure 12: Index distribution for the two age groups

As displayed in Figure 10, all approaches assimilate this initial purchase distribution and mainly recommend articles from the indexes *Lingeries/Tights*, *Ladieswear*, *Divided*, *Ladies Accessories*, and *Sport*. In particular, the GNN model results in an extreme exposure distribution of articles per index, with the exposure of the index *Lingeries/Tights* being four times higher than the second-highest index exposure. The ranking of the index exposure is also present for the Random Walk approach. However, the overall exposure distribution is not as extreme as for the GNN. For the FastRP & KNN method, the exposure of the indices is evenly distributed and therefore exhibits the highest item fairness.
To conclude, we saw that none of the approaches achieved good performance for every aspect of our evaluation strategy. While the FastRP & KNN strategy maintains a high catalog coverage and an even index exposure distribution, it fails at correctly recommending future purchases. In opposition, the GNN approach achieves a high MAP@10 value and ensures high customer fairness regarding both aspects, age, and cold start. A possible reason for this performance is a strong bias towards popular items, which results in an extremely low catalog coverage and uneven index exposure. In this first implementation, both of our approaches can not simultaneously optimize the quality of the recommendations and the item-sided fairness. However, we hypothesize both strategies exhibit the potential to do so if their specific limitations are the focus of future enhancements.

5 Conclusion and Future Work

In this project, we created a fashion graph-based recommender system using H&M Kaggle dataset[4]. Three approaches were implemented, namely Random Walk, Graph Embedding (FastRP & KNN), and Graph Neural Networks (GNN). Their evaluation focused on how relevant and fair the generated recommendations were.
It is worth mentioning that one key factor for the success of all approaches is the graph architecture. In future steps, different structures can be tested, for example, a purely homogeneous graph, removing certain nodes/edges or even including more knowledge about trends and customer preferences. A different graph topology might heavily impact the performance of our strategies.
For the Graph Embedding approach, we noticed a performance decrease compared to the Random Walk approach. In order to possibly see a better performance, one can use the FastRP algorithm with an embedding dimension of 256 instead of 32. On increasing the embedding dimension, not only FastRP but also the KNN part increases their memory requirements and runtime linearly. Therefore, changing the embedding dimension to a higher value would require a database server with high configuration.
During the implementation and training phase of the GNN approach, we observed that the recommendations were diverse for customers with long purchase histories and no significant pattern in the bought articles. At the same time, those customers received better recommendations when only 1% of data was used to train the GNN model. Our assumption is that a smaller purchase history might better reflect the current taste of the customer and fashion trends based on seasonality. To further explore this theory, one can add a time-dependent weight to purchases while training. This would give lower importance to older purchases and higher importance to recent ones enabling them to have better chances of reflecting the current customer preferences.
In addition to this, we did not explore deeply parameter optimization, also called fine-tuning, for the GNN approach. As a future step, one can optimize the hyperparameters and note down their findings on how the quality of the generated recommendations differs with the change in parameters. These modifications can include changing the loss function, learning rate, activation function, or the number of layers. One can also perform experiments on how the recommendation varies when a different graph convolution is used instead of GraphSAGE.
For the scope of this project, our focus was to evaluate the fairness of the generated recommendations. To go one step further, one can implement the notion of fairness in these recommender systems by reordering or removing articles in the recommendations list depending on the area of focus. For instance, if we want to implement item-sided fairness, which means that not only the popular items are recommended, one would like to get recommendations for new items as well to include the factor of novelty. Besides, in the case of GNN, the intent to implement fairness can be incorporated during negative sampling considering item popularity instead of random setting.
We observed that the GNN probably suffers from popularity bias when we compared the catalog coverage of the approaches. To identify this bias, the effective catalog size as a metric would be better since it also measures how often specific items are recommended in the catalog[7]. After identifying the popularity bias for the GNN, one can rectify the bias by implementing a candidate retrieval mechanism, so that edge probabilities are

calculated against a balanced and diverse set of articles and not the entire catalog which is not equally distributed in article types, see Figure 2[20]. Thus, we can improve the coverage and index exposure for the GNN model.
From the evaluation perspective, we have evaluated MAP@K and Catalog Coverage to get insights into the overall relevancy of the recommendations generated and the percentage of the catalog articles that were actually recommended. In order to acquire better insights into the strengths and weaknesses of the approaches, more metrics could be evaluated. For instance, it could be the case that an approach such as FastRP & KNN, focusing on the similarity factor of customer and article, might not get sufficient hits to contribute to a higher value of MAP@K but could perform better in intra-list similarity measure i.e. the similarity between articles in the recommended list.
More specifically, one should evaluate the recommender system with test users, called online A/B experiments. Research shows that the relations between offline, online, and user evaluation results are not linear and sometimes contradict each other[1]. Online evaluation is, therefore crucial part of quality assessment. In such a real-life scenario, the system provides a list of recommended articles to end users, and, if they are relevant, they can buy these articles. Using this purchase data, one could calculate a more accurate MAP@10 based on the users' experience and actual interaction with the recommendations. However, we cannot perform this evaluation and therefore are limited to calculating the MAP@10 with purchases already made.
Additionally, one could try to derive information on the catalog for different seasons. Availability of H&M articles might be restricted to certain periods, which influences the shopping behavior of customers, leading to an exposure bias in the data set and ultimately compromising evaluation metrics such as MAP@K or Catalog Coverage. Particularly, in the case of the GNN, more sophisticated architectures can be used to prevent this exposure bias, such as Temporal Graph Neural Network (TGN). In the current setup, we learn embeddings of customers and articles, such that we can predict accurately a purchase probability. With TGNs we could make this embedding generation time-dependent[15], preventing the recommendation of articles that might not be in the catalog during the evaluation period.
Further, depending on the stakeholder's requirements, different metrics could be selected for evaluation, in order to choose the approach with better performance in the metric we want to prioritize.

Bibliography

References

[1] Joeran Beel and Stefan Langer. "A comparison of Offline Evaluations, Online Evaluations, and User Studies". In: Dec. 2014.

[2] Haochen Chen et al. *Fast and Accurate Network Embeddings via Very Sparse Random Projection*. 2019. arXiv: 1908.11512. URL: http://arxiv.org/abs/1908.11512.

[3] Yashar Deldjoo et al. *A Review of Modern Fashion Recommender Systems*. 2022. DOI: 10.48550/ARXIV.2202.02757. URL: https://arxiv.org/abs/2202.02757.

[4] Carlos García Ling et al. *H&M Personalized Fashion Recommendations*. 2022. URL: https://kaggle.com/competitions/h-and-m-personalized-fashion-recommendations.

[5] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. "Beyond Accuracy: Evaluating Recommender Systems by Coverage and Serendipity". In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10. Barcelona, Spain: Association for Computing Machinery, 2010, 257â€"260. ISBN: 9781605589060. DOI: 10.1145/1864708.1864761. URL: https://doi.org/10.1145/1864708.1864761.

[6] David Goldberg et al. "Using Collaborative Filtering to Weave an Information Tapestry". In: *Commun. ACM* 35.12 (Dec. 1992), 61â€"70. ISSN: 0001-0782. DOI: 10.1145/138859.138867. URL: https://doi.org/10.1145/138859.138867.

[7] Carlos A. Gomez-Uribe and Neil Hunt. "The Netflix Recommender System: Algorithms, Business Value, and Innovation". In: *ACM Trans. Manage. Inf. Syst.* 6.4 (Dec. 2016). ISSN: 2158-656X. DOI: 10.1145/2843948. URL: https://doi.org/10.1145/2843948.

[8] William L. Hamilton, Rex Ying, and Jure Leskovec. *Inductive Representation Learning on Large Graphs*. 2017. DOI: 10.48550/ARXIV.1706.02216. URL: https://arxiv.org/abs/1706.02216.

[9] Bhushan Kotnis and Vivi Nastase. *Analysis of the Impact of Negative Sampling on Link Prediction in Knowledge Graphs*. 2017. DOI: 10.48550/ARXIV.1708.06816. URL: https://arxiv.org/abs/1708.06816.

[10] Jurij Leskovec. *Lecture CS224W: Machine Learning with Graphs*. Stanford University, 2021. URL: https://web.stanford.edu/class/cs224w/.

[11] Yunqi Li et al. *Fairness in Recommendation: A Survey*. 2022. DOI: 10.48550/ARXIV.2205.13619. URL: https://arxiv.org/abs/2205.13619.

[12] Claire Longo. *Evaluation Metrics for Recommender Systems*. 2018. URL: https://towardsdatascience.com/evaluation-metrics-for-recommender-systems-df56c6611093.

[13] Zaiqiao Meng et al. “Exploring Data Splitting Strategies for the Evaluation of Recommendation Models”. In: *Proceedings of the 14th ACM Conference on Recommender Systems*. RecSys ’20. Virtual Event, Brazil: Association for Computing Machinery, 2020, 681â€“686. ISBN: 9781450375832. DOI: 10.1145/3383313.3418479. URL: https://doi.org/10.1145/3383313.3418479.

[14] Lawrence Page et al. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report 1999-66. Previous number = SIDL-WP-1999-0120. Stanford InfoLab, Nov. 1999. URL: http://ilpubs.stanford.edu:8090/422/.

[15] Emanuele Rossi et al. *Temporal Graph Networks for Deep Learning on Dynamic Graphs*. 2020. DOI: 10.48550/ARXIV.2006.10637. URL: https://arxiv.org/abs/2006.10637.

[16] Michael Schlichtkrull et al. *Modeling Relational Data with Graph Convolutional Networks*. 2017. DOI: 10.48550/ARXIV.1703.06103. URL: https://arxiv.org/abs/1703.06103.

[17] Shoujin Wang et al. “Graph Learning based Recommender Systems: A Review”. In: Apr. 2021. DOI: 10.24963/ijcai.2021/630.

[18] Le Wu et al. *Learning Fair Representations for Recommendation: A Graph-based Perspective*. DOI: 10.48550/ARXIV.2102.09140. URL: https://arxiv.org/abs/2102.09140.

[19] Rex Ying et al. “Graph Convolutional Neural Networks for Web-Scale Recommender Systems”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, July 2018. DOI: 10.1145/3219819.3219890. URL: https://doi.org/10.1145%2F3219819.3219890.

[20] Han Zhang et al. *Towards Personalized and Semantic Retrieval: An End-to-End Solution for E-commerce Search via Embedding Learning*. DOI: 10.48550/ARXIV.2006.02282. URL: https://arxiv.org/abs/2006.02282.