Chair of Computational Modeling and Simulation
TUM School of Engineering and Design
Technical University of Munich

AI-based Next Command Prediction Through BIM Log Files

Scientific work to obtain the degree

Master of Science (M.Sc.)

at the TUM School of Engineering and Design
of the Technical University of Munich.

Supervised by	Prof. Dr.-Ing. André Borrmann
	Dr. Stavros Nousias
	Changyu Du
	Chair of Computational Modeling and Simulation
Submitted by	Omar Elsaka
	e-Mail:
Submitted on	December 24, 2024

Abstract

Sequential recommendation systems are created to predict the next item a user will likely interact with using temporal patterns and contextual information learned from historical user-item interactions. This study evaluates a Dynamic Graph Neural Network for Sequential Recommendation (DGSR) framework that attempts to effectively model user preferences to predict the next BIM command.

The proposed method first trains a DGSR model on existing user-command interactions to generate embeddings that capture sequential and structural information. For new, unseen users, a similarity-based weighted average aggregation is introduced to transfer embeddings from the pretrained user nodes to the new nodes based on their initial interactions. These aggregated embeddings are used to infer new user predictions. The model is then partially retrained to boost the new users' representations.

This hybrid approach combines the strengths of pretrained embeddings with adaptive retraining, effectively bridging the gap between transductive methods and production environments. Evaluations on a BIM log file dataset show the model's ability to predict the next user commands. The findings have shown the potential of dynamic graph-based recommendation systems in niche domains, providing a robust solution for sequential prediction tasks.

Zusammenfassung

Sequentielle Empfehlungssysteme werden erstellt, um das nächste Element vorherzusagen, mit dem ein Benutzer wahrscheinlich interagieren wird, indem zeitliche Muster und kontextuelle Informationen aus historischen Benutzer-Element-Interaktionen gelernt werden. Diese Studie evaluiert ein Dynamic Graph Neural Network for Sequential Recommendation (DGSR) Framework, das versucht, Benutzerpräferenzen effektiv zu modellieren, um den nächsten BIM-Befehl vorherzusagen.

Die vorgeschlagene Methode trainiert zunächst ein DGSR-Modell auf bestehenden Benutzer-Befehls-Interaktionen, um Einbettungen zu generieren, die sequenzielle und strukturelle Informationen erfassen. Für neue, unbekannte Benutzer wird eine auf Ähnlichkeit basierende gewichtete Durchschnittsaggregation eingeführt, um Einbettungen von den vortrainierten Benutzerknoten auf die neuen Knoten zu übertragen, die auf ihren ursprünglichen Interaktionen basieren. Diese aggregierten Einbettungen werden verwendet, um neue Benutzerprognosen abzuleiten. Das Modell wird dann teilweise neu trainiert, um die Repräsentationen der neuen Benutzer zu verbessern.

Dieser hybride Ansatz kombiniert die Stärken von vortrainierten Einbettungen mit adaptivem Retraining und überbrückt so effektiv die Lücke zwischen transduktiven Methoden und Produktionsumgebungen. Auswertungen eines BIM-Logfile-Datensatzes zeigen, dass das Modell in der Lage ist, die nächsten Benutzerbefehle vorherzusagen. Die Ergebnisse haben das Potenzial dynamischer graphbasierter Empfehlungssysteme in Nischendomänen aufgezeigt, die eine robuste Lösung für sequenzielle Vorhersageaufgaben bieten.

Contents

List of Figures

List of Tables

List of Abbreviations

BIM Building Information Modeling

CAD Computer-Aided Design

AEC Architecture, Engineering, and Construction

DSGR Dynamic Graph Neural Networks for Sequential Recommendation

RNN Recurrent Neural Network

LSTM Long short-term memory

GNN Graph Neural Network

MF Matrix Factorization

SVD Singular Value Decomposition

ADAM Adaptive Moment Estimation

DNN Deep Neural Networks

GMF Generalized Matrix Factorization

MLP Multi-Layer Perceptron

ReLU Rectified Linear Unit

GCN Graph Convolutional Network

GAT Graphic Attention Network

CNN Conolutional Neural Network

NGCF Neural Graph Collaborative Filtering

NCF Neural Collaborative Filtering

ML Machine Learning

CF Collaborative Filtering

Chapter 1

Introduction

1.1 Background

Approximately two decades ago, construction projects were designed using Computer-Aided Design (CAD) software instead of traditional drafting tools, such as rulers and pencils. This transition resulted in higher precision and faster generation of drawings. CAD software allowed designers to assign objects to groups and layers, which facilitated coordination among project stakeholders (RUSSEL and ELGER, 2008). Although the utilization of CAD tools produced drawings much faster and more efficiently, it was still slow because an engineer had to draw one drawing at a time. In addition, any small change to any part had to be manually applied in all drawings in which this part appears (RADNIA, 2021). Furthermore, integrating drawings from different disciplines using CAD software was challenging and prone to significant errors (BORRMANN et al., 2018).

To overcome this limitation, the Architecture, Engineering, and Construction (AEC) industry has begun to adopt Building Information Modeling (BIM) from the design phase until monitoring and facility management. BIM is typically defined as the process of creating, utilizing, and managing digital representations across the whole life cycle of a project. BIM software facilitates the continuous changes that usually occur in the design and planning phase without causing significant data losses. One of the principal advantages of BIM over CAD is the automatic updating of two-dimensional drawings in response to any change in the three-dimensional model. By leveraging computer technology, BIM enables the storage, maintenance, and exchange of information through detailed digital representations (BORRMANN et al., 2018).

That's why there has been a huge transition from CAD to BIM in recent years. Although several countries have adopted BIM for more than 15 years, like the USA, BIM is not yet established in the entire industry in Germany. It has been gradually introduced in infrastructure projects in Germany since 2020, as shown in Figure 1.1. Many public and private owners either demand BIM or have already adopted it for several years, such as Siemens Real Estate, which published its own BIM Standard Vision in 2017. In addition, a survey conducted in 2021 revealed that 46% of BIM users in Germany apply BIM to 50% or more of their projects. It is anticipated that the number of BIM users will increase to 64% within two or three years. Approximately three-quarters of German BIM users are designers (including architects, engineers, and consultants), whereas about one-quarter are contractors (JONES & LAQUIDARA-CARR, 2021). The results of the survey demonstrate that the majority of instances of adoption occur during the design phase.

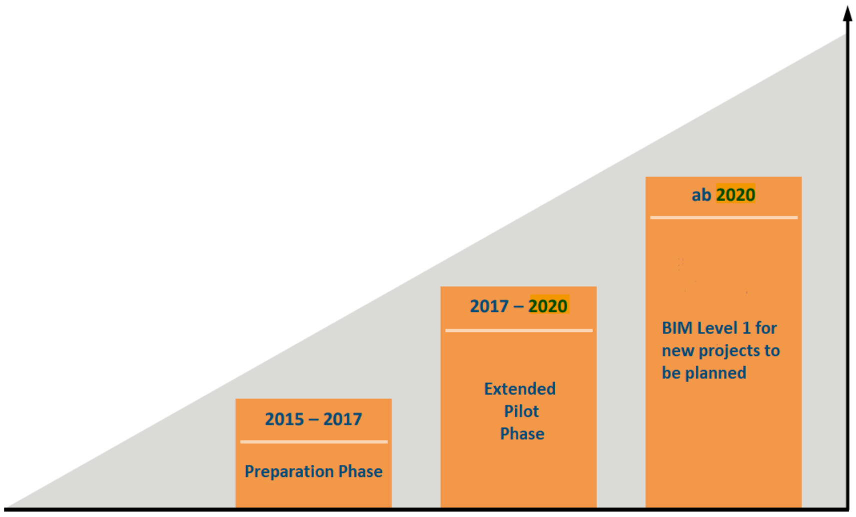


Figure 1.1: Phased introduction of BIM in Germany (BUNDESMINISTERIUM FÜR VERKEHR UND DIGITALE INFRASTRUKTUR, 2015)

BIM helps designers create semantically enriched and digital multi-dimensional models with parametric objects using object-oriented modeling software (e.g., Revit, Vectorworks, Archicad). During the design process, designers and engineers carry out repetitive and mechanical tasks that must be done manually and are laborious and time-consuming. With mechanical tasks comes lots of errors and rework. According to (OYEWOBI & OGUNSEMI, 2010), rework is "the waste or redundant part of a project that has become part of the construction process." During the design phase, architects and engineers dedicate most of their time to implementing modifications or rectifying errors that arise. This leads to 25% of cost overruns and 70% of schedule delays (ANDREW SHING-TAO CHANG, 2002). This renders the AEC industry one of the industries with the least efficiency and productivity (FULFORD & STANDING, 2014).

Furthermore, the impact of natural forms on architectural design has prompted a transition from Euclidean geometry to free-form, non-linear shapes (Brebbia & Collins, 2004). This has introduced an additional layer of complexity to the modeling process, with different building elements transitioning from standard shapes such as squares and circles to more intricate shapes. As a result, designers are required to possess a high level of proficiency in utilizing the full range of tools available in a BIM software package, which is, in turn, continuously updated with new tools. Although their original goal is to facilitate the modeling process, they require the modeler to always be updated with newly released tools. This introduces further complexity to the modeling process, which requires a time-consuming and financially burdensome regular training for designers and engineers.

Aligning with (van der Aalst, 2016) description of a process, BIM modeling consists of several cases, each of which is formed by a series of events that share typical attributes, including timestamp, activity, actor, and others. When designers or modelers encounter an issue during modeling, they retrieve past knowledge and experiences, either online, in older projects, or from more experienced colleagues. However, given the structure of the design process and the division of work among multiple designers, it would be unproductive for a designer to consistently seek assistance from a colleague. This highlights the necessity of leveraging the capabilities of BIM as a central repository for design information. It automatically saves comprehensive records of all users' modeling activities with timestamps, designer-software interactions, and system information into the expanding volumes of design event logs (Pan & Zhang, 2020a). Event logs are characterized by several attributes. They are sequential in nature, with recorded events following a specific order. They are typically goal-oriented and belong to a specific functional context.

Furthermore, they change over time due to various factors, including business needs and technological advances (van der Aalst, 2016). In other words, event logs are semantically rich data gathered automatically without human intervention (Pan, 2021). They are suitable sources for process mining, whereby all modelers' historical knowledge and experience base are recorded.

Design logs, which capture detailed process-specific information to continuously monitor modeling activities, are also analogous to web server logs, which automatically record the sequence of user activities. Various data mining techniques have been applied to web server logs to reveal hidden information about users' navigational behavior, a field known as web usage mining. For instance, by anticipating a user's perspective request or action through analyzing weblogs, recommendation systems, and tailored web content can be developed to align with users' preferences and enhance their experience (SRIVASTAVA et al., 2000).

Another comparable domain is e-commerce. Despite the widespread and rapid growth of e-commerce sales, the online conversion rates rarely exceed 5% (EMARKETER, 2014). Online business stakeholders have always struggled to convert a shopping filled with items into actual sales (EMARKETER, 2014). That's why tracking customers' behavioral aspects is one of the primary advantages of online businesses. Collecting and mining this data and the later application of recommender systems helped revolutionize this field. Research from McKinsey & Company has found that product recommendations can significantly contribute to higher sales figures and increase conversion rates by up to 300& (COMPANY, 2020).

Driven by data availability in many domains and enhanced hardware capabilities represented by massive storage and GPU, ML has experienced rapid growth in recent years. These sophisticated techniques are now more accessible due to the wide range of available open-source ML tools. ML is now widely applied to solve problems and automate routine tasks across various industries, especially in the field of recommender systems and sequential recommendation.

1.2 Motivation

The aforementioned successes would not have been achievable without the implementation of data collection and mining processes that record and analyze user behaviors and interests. Given the superior predictive performance of user behavior-based modeling on weblogs and historical shopping interests, there is reason to believe that BIM design event logs can be an effective data source to inform designers' decisions during the modeling process. As BIM is increasingly adopted, the data contained within event logs will reveal much hitherto unknown information, requiring further exploration (PAN & ZHANG, 2020a). However, this also poses certain challenges. For instance, an increasing volume of disorganized and non-intuitive data is accumulated automatically as the BIM application expands. It increases exponentially in the BIM platform, exhibiting characteristics of "Big Data." For example, the design data of an airport terminal with an area of 548,300 m^2 can reach 50 GB (LIN et al., 2016).

In their unprocessed form, the design log files are filled with uncertainty, subjectivity, and ambiguity. This necessitates arduous data manipulation to prevent sub-optimal outcomes, as evidenced by (PAN, 2021). This gives rise to two mutually interdependent challenges. People without a lot of BIM experience might find the large and complex data sets too overwhelming. This makes it hard to get the important information and key features. Also, if the data is inaccurate and not managed well, the quality of the data can get worse. This can lead to unreliable knowledge discovery and decision-making. As a result, there is a significant gap between those working with BIM data and those with expertise in data science (PENG et al., 2017).

In contrast to other sectors, the AEC industry has been relatively slow to adopt Machine Learning (ML) to address the challenges associated with BIM in construction. This lag can be attributed to several factors. On the one hand, AEC professionals often lack awareness of the potential benefits of ML or the expertise needed to integrate these techniques into their workflows. On the other hand, computer scientists and ML practitioners frequently have limited familiarity with the domain-specific requirements of BIM, such as its reliance on complex geometrical data, temporal dependencies, and the collaborative nature of construction projects.

This disconnect between disciplines has hindered developing and adopting solutions tailored to the unique needs of the AEC industry. While other fields—such as finance, healthcare, and manufacturing—have seen improvements driven by ML, applying these methods in construction remains in its early stages. Even existing studies that have attempted to leverage ML techniques in this domain often focus on narrow problems or exploratory use cases. As will be elaborated in Chapter 3, much of the current research has barely scratched the surface, leaving many opportunities for innovation untapped.

This introduction provides an overview of the challenges currently facing the AEC industry, largely due to the ever-increasing complexity and the increasing productivity of BIM authoring tools. Furthermore, it emphasizes the prospective advantages of employing the "big data" inherent in BIM event log files and the groundbreaking developments in machine learning algorithms to confront these challenges. Our objective is to capitalize on this knowledge base and cutting-edge machine learning algorithms to optimize the modeling process in the design phase, thereby enhancing efficiency and taking a further step towards design automation.

1.3 Research objectives

In this study, we aim to create a pipeline that utilizes BIM raw event log files to extract logical modeling sequences. We also aim to test the applicability of deep neural networks in BIM next command prediction, especially Graph Neural Network (GNN). Our goal in this research is to answer the following questions:

- How can a robust pipeline be designed to preprocess event logs from BIM authoring tools and extract meaningful modeling sequences for next-command prediction?
- How can GNNs be adapted to model sequential interactions in BIM environments for next-command prediction, and how do they compare with existing deep-learning methods?
- How can a similarity-based inference mechanism enable GNNs to generalize to unseen user graphs for next-command prediction in BIM?
- What is the impact of scalability on accuracy in dynamic graph recommendation systems for BIM?

By addressing these questions, this research aims to bridge the gap between the potential of BIM event log data and its practical applications in improving design workflows and decreasing reworks.

Chapter 2

Theoretical background

The sequential recommendation system has evolved to become one of the most crucial means of predicting user preference, with the aim of improving personalized experiences within e-commerce, entertainment, and content delivery domains. In essence, the interaction pattern of user sequences serves as the foundational element of these systems, utilizing insights from temporal dependencies and evolving user preferences for prediction to anticipate future actions. Theoretically, sequential recommendation techniques range from collaborative filtering to sequence modeling and graph-based learning.

This section looks at some of the very foundational concepts of sequential recommendation. It commences with an overview of collaborative filtering methods, highlighting how recent state-of-the-art has moved from the classical Matrix Factorisation (MF) approach towards current advanced deep learning-based approaches. Subsequently, it provides specific views on methodologies derived from sequence modeling, with particular reference to Recurrent Neural Networks (RNNs) and attention mechanisms for modeling temporal dependence in users' behaviors. Finally, the chapter ends with a discussion of recent advancements carried out in GNNs and their application for user-item interaction modeling, showcasing the emergence of dynamic models that learn the evolving user preferences.

2.1 Sequential Recommendation

Sequential recommendation tools have been developed to predict the subsequent item a user will interact with based on their historical sequence of interactions. Such systems are commonly utilized in various contexts, including e-commerce, streaming services, and social media platforms. The evolution of sequential recommendation models has its roots in recommender systems. In contrast to traditional recommender systems, which model the interactions as independent events, sequential recommendation considers the order in which users engage with different items, thereby capturing the evolution of interest over time. (Luo et al., 2024)

Although different sequential recommendation tools differ in their work, the following factors are the main players in formalizing the problem. According to (Boka et al., 2024) factors include user-item interaction, sequence modeling, and short and long-term preferences. User-item interaction models users' behaviors towards the items, such as clicks, views, purchases, and ratings. Sequence modeling is the technique of collecting the historical dependencies in user interactions and predicting similar future behavior. Lastly, the differentiation between short-term preferences, which consider the users' behavior within a session, and long-term preferences of the user that change over several sessions.

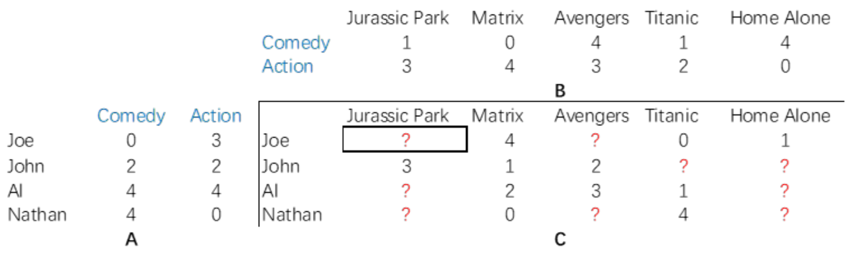
In the context of recommender systems, two fundamental approaches that all systems rely on are content-based filtering and collaborative filtering. Key components of these systems include user-item interactions, modeling of sequences, and the distinction between session-based and long-term preferences. Hence, they can be further classified into two categories: traditional techniques and methods based on neural networks. Traditional methods include MF and Markov chains, which are based on mathematical techniques of pattern uncovering. More recently, neural network-based approaches have been suggested to include deep learning, attention mechanisms, hybrid models, contrastive learning, and graph-based architectures that leverage complex patterns learned from data to improve the accuracy of recommendations. Using sophisticated data analysis and learning strategies, these evolving techniques enable a more fine-grained understanding of users' immediate and long-term preferences.

Various recommender system methods extend the same ideas, and many newer methods provide new twists or refinements to earlier methods. For example, traditional methods like MF and newer techniques like Dynamic Graph Neural Networks for Sequential Recommendation (DGSR) are closely related, with the latter advancing the former by refining how user and item representations are learned. This section provides an overview of the development of sequential recommendation, referencing in detail only the methods and concepts related to our methodology.

2.1.1 Content-based Filtering

In content-based filtering, recommendations are made using the characteristics or attributes of items based on previous preferences and interactions. It aims to recommend new but similar items to the user. The contrary to collaborative filtering, which depends on the patterns of similarity between users, content-based filtering operates independently of other users' data and relies exclusively on item-specific characteristics. This difference in approach enables the content-based systems to provide recommendations on a user's historical preferences without needing information from other users. (Ricci et al., 2010)

Figure 2.1 provides a simple example of content-based filtering. In this example, the task predicts missing values in a user-item rating matrix. Matrix A shows user genre preferences, Matrix B displays the genres of each movie, and Matrix C contains the interaction data for each item (user-item ratings). The goal is to estimate Joe's rating for Jurassic Park based on the available data in Matrices A and B. To calculate this, we can perform an element-wise multiplication of Joe's genre preferences from Matrix A with the movie genres in Matrix B and then normalize the result. For example, Joe's rating for Jurassic Park would be determined as follows: $(0 \times 1) + (3 \times 3)$, and then divided by 8 to ensure the ratings are within the range of ratings.



B

	Jurassic Park	Matrix	Avengers	Titanic	Home Alone
Comedy	1	0	4	1	4
Action	3	4	3	2	0

A

	Comedy	Action
Joe	0	3
John	2	2
Al	4	4
Nathan	4	0

C

	Jurassic Park	Matrix	Avengers	Titanic	Home Alone
Joe	?	4	?	0	1
John	3	1	2	?	?
Al	?	2	3	1	?
Nathan	?	0	?	4	?

Figure 2.1: Example of predicting a missing rating in a user-item matrix with content-based filtering (Koren et al., 2009)

In larger applications where more features are available, matrices A and B can be regarded as user and item profiles. The user's profile is constructed based on his tastes and activities, such as ratings, clicks, and minutes watched in movie trailers. Item's profile can contain features like actors and actresses, director, release year, and IMDb ratings. Those features are then represented by vectors. The recommendation can consequently take place by calculating the cosine similarity or the dot product between the vectors of movies.

The content-based filtering approach also offers the potential to provide explanations highlighting the content features responsible for recommending an item. This can enhance user confidence in the system's suggestions and offer insights into their own preferences. Additionally, a content-based method allows users to input initial subject information to assist the system. However, it is so simple because there are obviously more relevant features to a movie than genres only. Similarly, there are more features to a BIM command than its title.

2.1.2 Collaborative Filtering

Collaborative Filtering (CF) constitutes a further alternative method in which the underlying core idea relies on all users' collective assessments and behaviors to estimate individual users' interests. In other words, it operates under the assumption that users who have demonstrated a history of agreement in their assessments or ratings of items will likely exhibit a similar pattern in the future. By analyzing historical data, CF methods look for patterns or relationships among users and items to be able to predict a user's preferences without requiring details about the items themselves. (SCHAFER et al., 1999).

CF techniques could be divided into two classes: memory-based methods, which generate recommendations directly from the user-item interaction database, and model-based methods, which model ratings and interactions of users first and then make predictions using that model. (KOREN et al., 2021).

2.1.2.1 Memory-based Methods

Memory-based methods can be further divided into user and item-based collaborative filtering. They might be similar in how they are calculated but differ in the data each type uses to generate recommendations. For instance, first introduced by (RESNICK et al., 1994), user-based methods operate mainly on the user-user similarity principle. The fundamental algorithm operates in such a way as to ascertain the similarity between users, select a local neighborhood of users with similar tastes, and aggregate neighbor preferences in order to generate recommendations.

On the other hand, item-based methods, proposed by (SARWAR et al., 2001), focus on item-item similarity. This approach identifies items similar to those the user has liked in the past. The key steps include computing similarity between items, selecting a set of similar items, and combining the user's ratings on these similar items to predict ratings of new items. As asserted by (SARWAR et al., 2001), item-based methods frequently demonstrate superior scalability and enhanced quality of recommendations in comparison to user-based approaches, particularly within domains where items exhibit greater stability than user preferences over time.

Similarity computation is a critical step in both user-based and item-based CF. Common similarity measures include the Pearson correlation coefficient, which measures the linear relationship between two variables, defined as:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where x_i and y_i are the ratings of user/item x and y, respectively, and $\bar{x}$ and $\bar{y}$ are their mean ratings. Cosine Similarity measures the cosine of the angle between two vectors in a multi-dimensional space:

$$\cos(x, y) = \frac{x \cdot y}{||x|| \cdot ||y||}$$

Jaccard Similarity measures similarity between two sequences with no ratings, where we want to know common items that a pair of users have interacted with. Assuming that each user's sequence is a set of items, similarity is defined as the size of the intersection divided by the size of the union of two sets:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Once similarities are computed, a neighborhood of similar users or items is formed. Usually, this involves selecting the top-K most similar users/items or choosing all users/items above a certain similarity threshold. The size and quality of the neighborhood will have a great impact on the accuracy of the predictions (HERLOCKER et al., 1999). The final step is the aggregation of ratings of similar users or items to predict the ratings of new items. For user-based CF, a common prediction formula is:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N} sim(u,v) \cdot (r_{vi} - \bar{r}_v)}{\sum_{v \in N} |sim(u,v)|}$$

where $\hat{r}_{ui}$ is the predicted rating of user u for item i, $\bar{r}_u$ is the mean rating of user u, N is the set of neighbors, $sim(u,v)$ is the similarity between users u and v, and r_{vi} is the rating of neighbor v for item i (RESNICK et al., 1994). On the other hand, the prediction for the item-based method is computed as follows, where j represents items similar to item i(SARWAR et al., 2001):

$$\hat{r}_{ui} = \frac{\sum_{j \in N} sim(i,j) \cdot r_{uj}}{\sum_{j \in N} |sim(i,j)|}$$

Although memory-based methods have succeeded in many applications, they face several challenges. For instance, the user-item interaction matrix in many real-world scenarios is extremely sparse. So, if a user or item has few interactions, it's difficult to find reliable neighbors (ADOMAVICIUS & TUZHILIN, 2005). In addition, they face scalability issues as the number of users and items grows. They typically require storing and processing the entire user-item interaction matrix, which can be computationally expensive for large datasets (KOREN et al., 2009). Moreover, those methods are suited where input data reflects explicit user interests, such as ratings, while they struggle where only implicit user feedback is used as input.

2.1.2.2 Model-based methods

To overcome those limitations, model-based approaches use latent factors to explain ratings or preferences by identifying hidden patterns in how users and items interact (KOREN et al., 2009). They can also approximate user preferences from implicit feedback (where no explicit like or dislike signs are available). The model-based approach uses ML and data mining techniques to build predictive models based on the user-item interaction data. These methods train a model offline and then use the learned model to make predictions. The most widespread model-based approaches include MF and Markov chains, and recently, neural network and deep learning methods have gained wide popularity.

Matrix Factorization (MF) and Markov Chain

MF is one of the most widely used collaborative filtering techniques. It is a latent factor model that aims to decompose the large user-item interaction matrix into smaller, denser matrices of latent factors. The basic idea is the approximation of the rating as a dot product of user and item vectors (SALAKHUTDINOV & MNIH, 2007). Let's recall the example from Figure 2.1. The missing rating r_{ui} of the user u to the item i can be estimated as follows:

$$\hat{r}_{ui} = q_i^T p_u$$

This method typically uses Singular Value Decomposition (SVD) to identify latent factors within the available data (KOREN et al., 2009). However, due to the sparsity of the interaction matrix, conventional SVD yields are undefined when dealing with missing values in the matrix. As a result, (KOREN, 2008; PATEREK, 2007) suggested considering only recorded interactions and using regularization as a remedy for overfitting. The model is then typically learned by minimizing the regularized squared error on the set of known ratings:

$$\min_{q^*,p^*} \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2)$$

where K is the set of known ratings, r_{ui} is the rating of user u for item i, p_u and q_i are the latent factor vectors for user u and item i respectively, and λ is the regularization parameter. The equation is minimized using either stochastic gradient descent or alternating least squares.

One of the advantages of MF is that it is flexible and easily accommodates more data aspects. For instance, lots of users have systematic tendencies when rating items. These tendencies can be related to a specific type of item or are an innate characteristic of the user. These can be modeled as biases deviating from a user's average overall rating for all items as follows (KOREN et al., 2009):

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u$$

The estimated rating is divided into four parts: global average rating, item bias, user bias, and user-item interaction. Specific care should be given to modeling biases because they capture most of the observed interaction signal. Those models are, however, suited for static recommender systems, where the interactions' order or temporal dimension is not considered. In reality, users' inclinations evolve over time as new selections or trends emerge (KOREN et al., 2009). MF accommodates this change by modeling user factors as a function of time. The rating at time t is then computed as follows (KOREN, 2010):

$$\hat{r}_{ui}(t) = \mu + b_i(t) + b_u(t) + q_i^T p_u(t)$$

MF fails to capture the effect of the order of interactions on the next interaction. In this case, Markov Chain can be deployed to remedy this shortcoming (BOKA et al., 2024). The most popular techniques are the basic Markov Chain and latent Markov Chain. The basic technique leverages explicit observations to calculate probabilities; the latent technique computes the probabilities based on the Euclidean distance between the embedded Markov chains. Markov Chains are, however, limited to modeling short-term preferences, as they leverage the latest interaction or the few latest interactions to generate predictions. That's why they haven't been employed in sequential recommendation in past years (BOKA et al., 2024).

Although traditional model-based methods have succeeded greatly, they have also faced many limitations and challenges. For instance, in many real-world scenarios, the user-item interaction matrix is extremely sparse, leading to cold start problems for new users or items (ADOMAVICIUS & TUZHILIN, 2005). On top of that, memory-based methods tend to recommend popular items, potentially leading to a lack of diversity in recommendations. This problem can be seen where datasets include repeated interactions with a subset of items while the others have few interactions.

Consequently, advanced neural network methods have been developed to handle such challenges. These methods employ deep learning techniques to identify complex and non-linear patterns in user behavior and preferences. The next chapter covers those methodologies that have a substantial advancement in the domain of recommendation systems.

2.2 Deep Neural Networks

Neural networks have been designed to function analogously to the human brain. The human brain is layered and filled with interconnected neurons, with feedforward and feedback connections, thus allowing it to process information and construct complex representations of the sensory input. The most difficult functions that the human brain carries out are the cognitive functions, including language, abstract reasoning, and memory and learning. Such brain operations are the most difficult to define in terms of neural mechanisms. (Du & Swamy, 2019)

Inspired by this biological structure, DNNs consist of multiple layers of artificial neurons. Thus, one of the distinguishing characteristic features of DNNs, is the depth represented by many hidden layers. These allow the formation of representations of data in a very abstract way. Hence, DNNs can even capture detailed information from data, which enables them to achieve certain performances for tasks hitherto thought impossible by many. (Goodfellow, 2016)

In this chapter, several key techniques used in DNNs are explored, starting with Neural CF-one of the earlier approaches, which is static in nature and applied mainly to recommendation systems by the RNNs, which were developed to handle sequential data, such as time-series tasks and natural language processing. Subsequently, the Long Short-Term Memory (LSTM) network will be introduced as a subsequent innovation founded upon the principles of RNNs.

The next section will give a deeper overview of deep neural networks to understand how these methodologies work and what principles stand behind them. These networks will set a basis for acquiring basic notions necessary to learn the more advanced complex techniques.

2.2.1 Preliminaries on deep neural networks

A neuron is the basic processing unit in a neural network, acting as a node that processes incoming signals from other nodes. The first artificial neuron model was inspired by (McCulloch & Pitts, 1943). A mathematical simplification of this is as follows:

$$y = W \cdot X + b$$

where X represents the matrix of inputs comprising the features or variables being fed to the perceptron, W is the link weight matrix from the input that controls the magnitude of the influence exerted by each input feature on the output, b is a bias value or threshold that controls the signal. In order to learn the correct influence and threshold values, those weights and biases are then passed through a non-linear activation function.

Figure 2.2 illustrates the McCulloch-Pitts neuron model, the foundation for artificial neural networks. The inputs $x_1, x_2, \ldots, x_J$ are combined linearly through weights $w_1, w_2, \ldots, w_J$, summed at a node, and passed through an activation function $\varphi(\cdot)$. A threshold θ determines whether the neuron activates to produce the output y.

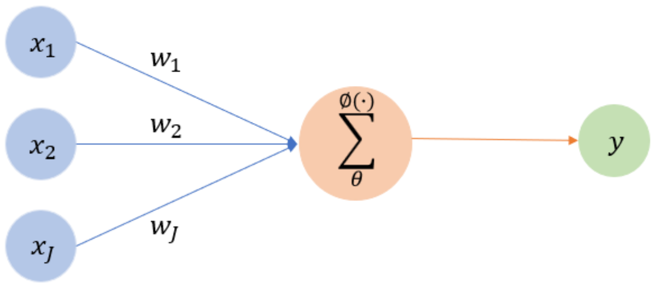


Figure 2.2: Simplified schematic of a neuron model according to (McCulloch & Pitts, 1943)

Learning is a key function in neural networks. A learning rule is an algorithm that determines the appropriate values for the weight matrix W and other network parameters. A training procedure for a neural network can be viewed as a nonlinear optimization problem - a search for the parameter values that minimize the cost function based on examples. This process is called the learning or training algorithm. Normally, neural networks are trained in epochs, wherein an epoch constitutes a complete cycle through which each training example once is presented to the network and processed using the learning algorithm. After training, the network captures complex relationships and acquires generalization capacity. A criterion is set to determine when training should stop to manage the learning process. (Du & Swamy, 2019)

The training of a neural network is feedforward propagation and backpropagation. Feedforward propagation is where input data is passed through the network, layer by layer, to compute an output or prediction. Every neuron does a weighted sum of its inputs at training, passing through some activation function that introduces nonlinearity. Ultimately, the output is compared with the expected one to calculate the error. Error is the deviation of the predicted value from the label value. Once the model has learned the weights and biases that best fit a particular problem through training, feedforward propagation computes the predictions for new data during inference without further modification, making it a fixed or "static" process. Feedforward neural networks, therefore, inherently have only static classification tasks. Their structure allows only the creation of a static mapping between inputs and outputs, which is adequate for tasks with no temporal dependencies. (STAUDEMEYER & MORRIS, 2019)

Backpropagation was developed to optimize the network performance by updating weights according to the error computed at every feedforward pass. The error will be backpropagated starting from the output layer, passing backward through the hidden layers towards the input layer. This is done by backpropagation through its iterative weight adjustment by calculating the error's gradient concerning each weight via the chain rule, enabling it to improve its predictions through adaptation to time-dependent tasks. A fundamental prerequisite for the backpropagation algorithm is the differentiability of each neuron's activation function. Consequently, the network can enhance its predictive capabilities through iterative refinement of its parameters, achieving an optimal solution by adjusting its weights. (STAUDEMEYER & MORRIS, 2019)

2.2.2 Neural Collaborative Filtering

Recently, neural network-based methods have been proposed as a flexible and powerful alternative to address some of the challenges in traditional model-based CF discussed in section 2.1.2.2. The prime motivation for using neural network-based methods in CF is its ability to capture and model complex nonlinear relationships intrinsic to user-item interactions. Hence, Neural Collaborative Filtering (NCF) is way more flexible and effective than classical methods, like MF, which normally can't capture intricate patterns from high-dimensional data. (HE et al., 2017)

The NCF framework architecture comprises many layers to represent y_{ui}, which is user-item interaction, as shown in Figure 2.3. In any neural architecture, the previous layer's output provides the feed for the next successive layer. The input layer represents the user u and item i as one-hot encoded sparse binary vectors, $\mathbf{v}_u^U$ and $\mathbf{v}_i^I$ respectively. Later, these vectors are passed via a fully connected embedding layer, transforming them into latent dense vector representations. In this manner, these embeddings learn to represent the intrinsic features of both users and items within one shared latent space. These latent vectors pass through many hidden neural network layers to better capture the complex nonlinear relationships between users and items. The output layer generates the predicted score denoted as $\hat{y}_{ui}$, and the model is usually trained by minimizing a pointwise loss function that measures the difference between $\hat{y}_{ui}$ and its ground truth value y_{ui}. (HE et al., 2017)

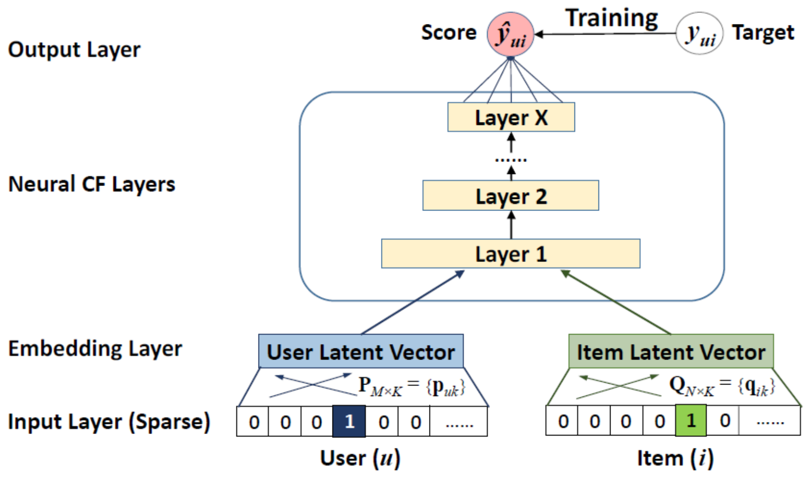


Figure 2.3: Neural Network-Based Collaborative Filtering Architecture (HE et al., 2017)

The predictive model in NCF can be expressed as: (HE et al., 2017)

$$\hat{y}_{ui} = f\left(\mathbf{P}^T\mathbf{v}_u^U, \mathbf{Q}^T\mathbf{v}_i^I \mid \mathbf{P}, \mathbf{Q}, \Theta_f\right),$$

where $\mathbf{P} \in \mathbb{R}^{M\times K}$ and $\mathbf{Q} \in \mathbb{R}^{N\times K}$ represent the latent factor matrices for users and items, respectively. Θ_f includes the parameters of the interaction function f.

The more generalized and extended variant of MF technique is the Generalized Matrix Factorisation (GMF). It is a NCF variant that combines strengths from traditional linear embedding and flexible neural networks. Specifically, GMF forms the base for several advanced NCF models by effectively combining the user and item representations on predictive tasks. The mapping function of the first NCF layer is hence defined as: (HE et al., 2017)

$$\phi_1(\mathbf{p}_u, \mathbf{q}_i) = \mathbf{p}_u \circ \mathbf{q}_i,$$

where $\circ$ denotes the element-wise product of the user and item latent vectors. This operation produces a combined latent representation. The resulting vector is then projected to the output layer, where the predicted rating $\hat{y}_{ui}$ is computed as:

$$\hat{y}_{ui} = a_{\text{out}}(\mathbf{h}^T(\mathbf{p}_u \circ \mathbf{q}_i)),$$

with a_{out} being the activation function and $\mathbf{h}$ the weights at the output layer.

An extension to the GMF model with the incorporation of a Multi-Layer Perceptron (MLP) is then introduced to capture richer, nonlinear interactions. Rather than solely utilizing element-wise operations, the MLP concatenates user and item embeddings, applying a series of nonlinear transformations. This architecture allows NCF to discern highly intricate relationships between users and items. Formally, this can be expressed as:

$$z_1 = \phi_1(\mathbf{p}_u, \mathbf{q}_i) = [\mathbf{p}_u; \mathbf{q}_i],$$

$$\phi_2(z_1) = a_2(W_2^T z_1 + b_2),$$

$$\vdots$$

$$\phi_L(z_{L-1}) = a_L(W_L^T z_{L-1} + b_L),$$

$$\hat{y}_{ui} = \sigma(\mathbf{h}^T \phi_L(z_{L-1})),$$

where W_x, b_x, and a_x represent the weight matrix, bias vector, and activation function of the x-th layer. The activation function may be sigmoid, hyperbolic tangent-tanh, or ReLU.

2.2.3 Recurrent Neural Network

While NCF is inherently static, which models interactions without considering the sequence in which they occur, Recurrent Neural Networks (RNNs) are particularly suited for handling sequential data. By retaining hidden states that help track information over time, they excel in tasks involving temporal or ordered patterns. (Boka et al., 2024)

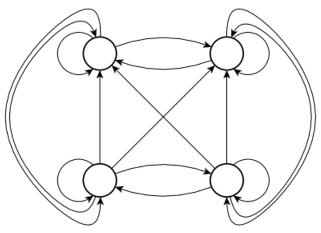


Figure 2.4: Feedback mechanism of RNNs (MIENYE et al., 2024)

Figure 2.4 illustrates the structure of a full RNN, highlighting its self-feedback connections. These connections enable the network to maintain an internal state across time steps. By looping back on themselves, they can carry information from previous time steps into the current processing stage, effectively allowing the network to 'remember' and process sequences of events over time (STAUDEMEYER & MORRIS, 2019). This architecture makes RNNs particularly effective for capturing temporal patterns in user behavior, enabling predicting upcoming interactions based on past ones. Their memory capability also proves valuable in tasks requiring contextual understanding, such as language processing and time-series forecasting (LUO et al., 2024)

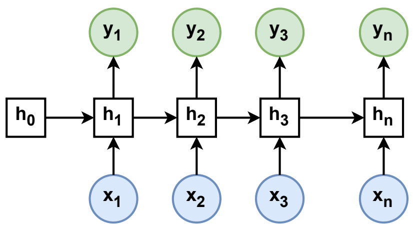


Figure 2.5: RNNs internal memory that maintains the internal state across time steps (MIENYE et al., 2024)

This extended functionality allows RNNs to incorporate both the current input X_t and the previous inputs $X_{0:t-1}$, as opposed to just the current input as in Feedforward Networks. The simplified model aggregates multiple hidden layers into a single Hidden Layer block H.

Mathematically, at time step t, the hidden state $H_t \in \mathbb{R}^{n \times h}$ is computed from the previous hidden state H_{t-1} and the current input X_t, using weight matrices W_{xh} and W_{hh}, along with a bias term b_h. The activation function φ (usually sigmoid or tanh) is applied to prepare the gradients for backpropagation. The output O_t is derived from the current hidden state. The following equations describe this process(SCHMIDT, 2019):

$$H_t = \varphi_h(X_t W_{xh} + H_{t-1} W_{hh} + b_h)$$

$$O_t = \varphi_o(H_t W_{ho} + b_o)$$

Since in RNNs, we need to propagate information through the recurrent connections in between steps, the most common and well-documented learning algorithms for training RNNs in these contexts are Backpropagation Through Time (BPTT), which propagates the error through time steps to update the weights. BPTT is a variant of backpropagation tailored for recurrent networks. By unfolding the RNN across its time steps, BPTT transforms the network into a feedforward-like structure where backpropagation can be applied to compute gradients and update the weights. The process involves calculating the loss function $L(O, Y)$ as the sum of the loss terms at each time step. (SCHMIDT, 2019)

$$L(O, Y) = \sum_{t=1}^{T} \ell_t(O_t, Y_t)$$

where $\ell_t(O_t, Y_t)$ is the loss term at time step t, and O_t and Y_t are the output and target values at time step t, respectively.

In the context of BPTT, the gradients are then computed with respect to three weight matrices: W_{xh} (input-to-hidden), W_{hh} (hidden-to-hidden), and W_{ho} (hidden-to-output). The partial derivatives with respect to these weights are calculated by applying the chain rule. The gradients are then used to update the weights during training.

BPTT is an offline learning method because the entire sequence is processed before the weights are updated. While effective at capturing long-term dependencies, it becomes unstable for long sequences due to accumulating very small or large gradient values. This issue arises because each hidden state H_t depends on all previous states, making the backpropagation through all time steps computationally expensive and prone to numerical instability. One possible method to solve this is Truncated BPTT, which restricts the backpropagation to a limited number of time steps, reducing the risk of gradient issues and making the computation more manageable. (SCHMIDT, 2019)

However, despite the benefits of Truncated BPTT, the problem of vanishing and exploding gradients still remains in practice. Standard RNNs might only focus on predicting the next item based on the most recent interaction, neglecting how past interactions might influence future choices. A more sophisticated variant of RNNs, known as LSTM networks, was introduced to overcome this challenge. They are designed to address these issues by incorporating memory cells that help retain and propagate information across long sequences, making them more effective at learning long-term dependencies. Unlike traditional RNNs, LSTM networks include mechanisms such as gates to regulate the flow of information, thereby mitigating the gradient problems associated with BPTT. Furthermore, they streamline the training process, balancing the strengths of offline and online learning methods while reducing computational complexity. (STAUDEMEYER & MORRIS, 2019)

2.3 Graph Neural Networks for Sequential Recommendation

While RNNs and their variants effectively capture temporal dependencies in sequential data, they are limited in handling sequences of variable sizes often present in real-world data. Graph-based models address this limitation by leveraging graph structures, where entities are depicted as nodes and their relationships as edges. Building on this foundation, Graph Neural Networks (GNNs) extend traditional graph-based models by incorporating learnable representations and iterative message-passing mechanisms. These advancements enhance the capacity of graph-based models to model complex relationships and provide accurate predictions, particularly in sequential recommendation scenarios. (CORSO et al., 2024; LUO et al., 2024).

2.3.1 Preliminaries on GNNs

The following section lays out the foundational components of GNNs, including their message-passing mechanisms, node update functions, and applications in sequential recommendation tasks.

Mathematically, a graph $G = (V, E)$ formally represents a set of connected objects called vertices (or nodes) V, where $V = \{v_1, v_2, ..., v_n\}$, and edges E, where each edge $e \in E$ connects a pair of nodes $e = (v_i, v_j)$. Edges can be directed or undirected, representing relationships or interactions between nodes and the flow of information between nodes. Figure 2.6 illustrates the two primary types of graphs. On the right, a directed graph is depicted, where edges have a specific direction, represented by arrows, indicating the flow of information or relationships between nodes. On the left, an undirected graph is shown, where edges do not have a direction, symbolizing bidirectional or mutual relationships between nodes.

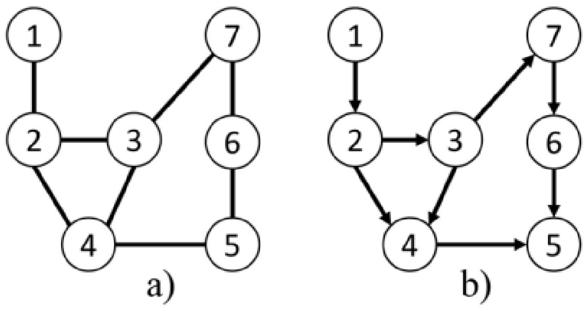


Figure 2.6: Directed and Undirected graphs (HERAKOVIC et al., 2019)

In a weighted graph, each edge $(i, j) \in E$ is associated with a weight w_{ij}. The graph structure is typically represented using an adjacency matrix $A \in \mathbb{R}^{|V|\times|V|}$, where each value represents the connectivity status between node i and j, where $A_i, j = 1$ if there is an edge between nodes i and j , and $A_i, j = 0$ otherwise. (CORSO et al., 2024)

To make this data machine-readable, each node v_i is initially transformed into a vector representation $h_i^{(0)}$. The exponent 0 indicates the input before the first neural network layer. Different types of nodes are assigned unique high-dimensional embeddings that reflect their distinct characteristics. Similarly, edges can have vector representations e_{ij} that capture attributes such as the type of connection between two nodes (e.g., single or double bond). (CORSO et al., 2024)

In GNNs, the central process involves iteratively aggregating and updating node representations through learnable functions, enabling the network to capture higher-order dependencies and complex relationships within the graph. The message-passing layer is key to this process, where nodes receive vector-based messages from their neighbors. The information gathered from neighboring nodes is aggregated into a single vector and used to update the representation of each node. Over multiple layers, this process allows nodes to refine their representations and capture increasingly complex patterns. (CORSO et al., 2024)

The aggregation of messages is commonly done through vector summation, and the updated representation for each node is then computed using a feedforward neural network, which applies a learnable linear transformation $W^{(\ell)}$ followed by a non-linearity like the ReLU activation function. Mathematically, the update rule for the representation $h_i^{(\ell)}$ of node i at layer ℓ can be written as (CORSO et al., 2024):

$$h_i^{(\ell+1)} = \text{ReLU}\left(W^{(\ell+1)}\left(h_i^{(\ell)} + \sum_j h_j^{(\ell)}\right)\right)$$

After the final message-passing layer, the representations of individual nodes are typically aggregated and transformed to make task-specific predictions. The approach to this aggregation depends on the nature of the task, as different problems may require outputs at different levels of granularity. (CORSO et al., 2024)

While GNNs have demonstrated considerable success across various applications, they still face notable limitations. Issues such as poor generalization to diverse real-world data, challenges with interpretability, and difficulties in identifying certain graph patterns highlight areas for improvement. Additionally, GNNs struggle with scalability to large graphs and over-smoothing, where node representations become indistinguishable after several layers. Over the past decade, numerous GNN variants have been proposed to address these challenges and cater to different tasks and graph structures. These architectures are generally classified based on the aggregation and transformation functions they employ. For instance, Graph Convolutional Networks (GCNs) use convolutional operations to capture local neighborhood information, while Graph Attention Networks (GATs(focus on enhancing the aggregation process and reducing computational bottlenecks (CORSO et al., 2024; VRAHATIS et al., 2024).

2.3.1.1 Graph Convolutional Networks

The Graph Convolutional Network (GCN) represents a foundational paradigm for GNNs. The principal objective of this architectural approach is to develop a computationally efficient version of CNN for graph-based data. GCNs can leverage the graph structure and aggregate node information from the neighborhoods in a convolutional fashion. They possess great expressive power to learn graph representations and achieve superior performance in various tasks and applications. (Vrahatis et al., 2024)

Each node $v_i \in V$ may have an associated feature vector $x_i \in \mathbb{R}^d$, where d represents the dimensionality of the feature space. The core idea of GCNs is to update each node's representation by combining its own features with the features of its neighbors, weighted by the graph structure. The learning process in the networks happens across multiple layers. At each layer, the node features x_i are updated by aggregating information from the node's neighbors, and these new representations are then used as input for subsequent layers. For the k-th graph convolution layer, the input node representations of all nodes are denoted by the matrix $\mathbf{H}^{(k-1)}$, and the output node representations are $\mathbf{H}^{(k)}$. Initially, the node representations are simply the original input features (F. Wu et al., 2019):

$$\mathbf{H}^{(0)} = \mathbf{X}$$

This process is repeated in subsequent layers, allowing the model to learn more complex and informative node representations progressively. The final output of the GCN is then used as input for downstream tasks, such as node classification or link prediction.

A GCN with multiple layers operates similarly to a MLP, where the feature vector of each node is updated across layers. However, in a GCN, the node's representation is updated by aggregating information from its neighbors at each layer. This process typically consists of three main stages: propagating the features across the graph, applying a linear transformation to the aggregated features, and then applying a non-linear activation function to introduce complexity in the representation. (F. Wu et al., 2019)

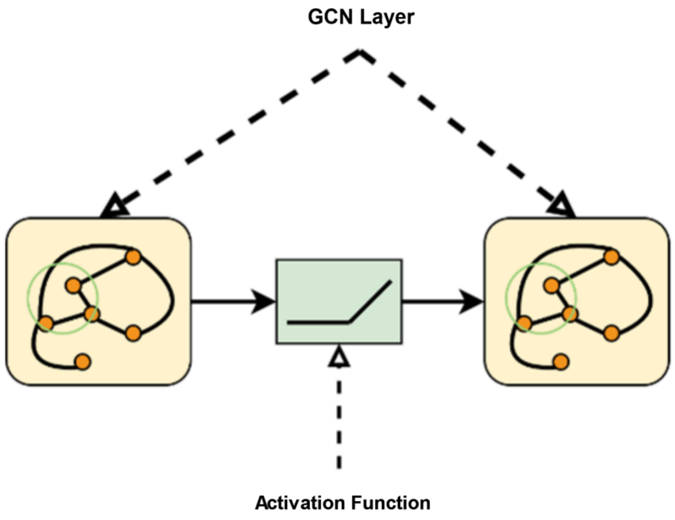


Figure 2.7: Illustration of GCN layers. Unlike traditional GNN, GCNs integrate node degrees to facilitate efficient normalization during the aggregation phase. (VRAHATIS et al., 2024)

2.3.1.2 Graph Attention Networks

While GCNs are effective in learning node representations by aggregating features from neighboring nodes, one limitation is that they treat all neighbors equally during the aggregation process, as GCNs do not have a mechanism to weigh the significance of each neighbor. This approach assumes that all neighbors contribute equally to the target node's representation, which can be problematic in graphs where certain neighbors play a more critical role than others.

To address this limitation, GATs (VELIČKOVIĆ et al., 2018) introduce a dynamic attention mechanism to the aggregation process, representing a transformative evolution in the field of GNNs, building on the foundations laid by GCNs. The core innovation of Graph Attention Networks (GATs) lies in their ability to assign weights to neighboring nodes during aggregation based on their relevance. This mechanism enables the network to dynamically assess and prioritize the significance of adjacent nodes, focusing more on relevant connections. This learned prioritization enhances the model's capacity to handle varying input sizes and improves performance in graphs where certain nodes play critical roles. (CORSO et al., 2024) or/and (VRAHATIS et al., 2024)

Unlike GCNs that normalize based only on node degrees, GATs learn attention weights that consider both the number of connections and how relevant node features are to each other. This adaptive weighting system represents a departure from the static approach of GCNs, allowing GATs to more efficiently model contextual significance within the graph. (CORSO et al., 2024; VRAHATIS et al., 2024)

Initially, each node is subjected to a linear transformation using a learnable weight matrix, denoted as $\mathbf{W}$. This transformation prepares the data for further processing and introduces a level of abstraction in the node representations. After this transformation, the model calculates attention coefficients representing the non-normalized attention weights between neighboring nodes. Specifically, for two connected nodes i and j, their embeddings $\mathbf{z}_i$ and $\mathbf{z}_j$ are concatenated, forming a combined vector $[\mathbf{z}_i \| \mathbf{z}_j]$. This concatenated vector is then subjected to a dot product with a learnable weight vector $\mathbf{a}$, effectively integrating the node features into the attention mechanism. To introduce non-linearity into the model, the LeakyReLU activation function is applied to the result of this dot product operation, mathematically expressed as (VRAHATIS et al., 2024):

$$e_{ij} = \text{LeakyReLU}(\mathbf{a}^\top [\mathbf{z}_i \| \mathbf{z}_j]),$$

where e_{ij} represents the raw attention coefficient. To ensure consistency and comparability across the attention scores of all neighboring nodes, the coefficients are normalized using the softmax function:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})},$$

where α_{ij} is the normalized attention coefficient, and $\mathcal{N}_i$ represents the set of neighbors of node i. This normalization ensures that the attention scores are scaled appropriately and sum to one for each node.

During the aggregation phase, the model combines embeddings from neighboring nodes, guided by the attention weights calculated in the previous step. However, GATs can address the potential instability of self-attention mechanisms by applying the concept of multi-head attention. This approach employs multiple attention mechanisms, referred to as "heads," each with unique parameters. These heads operate independently and in parallel, which increases the model's robustness and capacity to capture diverse features.

Each attention head produces its own output, subsequently integrated into the final representation. This process consolidates multiple perspectives into a unified output, enhancing the overall model performance. The mathematical relation for computing vector representations of nodes can be expressed as (VRAHATIS et al., 2024)

$$\mathbf{h}_i = \sum_{j \in \mathcal{N}(i)} \alpha_{ij} \mathbf{W} \mathbf{h}_j$$

2.3.2 Neural Graph Collaborative Filtering

While GCNs and GATs excel at graph-based tasks, they don't inherently encode collaborative signals from user-item interactions. Neural Graph Collaborative Filtering (NGCF) (X. WANG et al., 2019b) addresses this limitation by explicitly incorporating these signals through high-order connectivity propagation between users and items.

NGCF's architecture comprises three key components: an embedding layer for initial user-item representations, multiple propagation layers that refine embeddings through message-passing from neighbors, and a prediction layer that combines refined embeddings to compute user-item affinity scores. (X. WANG et al., 2019b)

The NGCF framework uses an embedding layer to represent users u and items i as vectors $\mathbf{e}_u$, $\mathbf{e}_i \in \mathbb{R}^d$ in a shared latent space stored in a parameter matrix $\mathbf{E}$. Unlike MF or NCF models, NGCF refines these embeddings through propagation layers using graph-based message passing, consisting of message construction and aggregation. For a user-item pair (u, i), messages pass from item to user through this framework:

$$\mathbf{m}_{u \leftarrow i} = \frac{1}{\sqrt{|N_u||N_i|}} \left(\mathbf{W}_1 \mathbf{e}_i + \mathbf{W}_2 (\mathbf{e}_i \odot \mathbf{e}_u) \right),$$

where $\mathbf{e}_i$ and $\mathbf{e}_u$ are the embeddings, $\mathbf{W}_1$ and $\mathbf{W}_2$ are trainable weights, and $\odot$ denotes element-wise multiplication. Aggregation combines messages from all neighbors N_u to refine user embeddings, where self-connections retain original embedding information. Higher-order propagation extends this process to l-hop neighbors, recursively aggregating messages to capture complex collaborative signals. The l-th layer representation for a user u is:

$$\mathbf{e}_u^{(l)} = \text{LeakyReLU}\left(\mathbf{m}_{u\leftarrow u}^{(l)} + \sum_{i\in N_u} \mathbf{m}_{u\leftarrow i}^{(l)}\right),$$

with $\mathbf{m}_{u\leftarrow i}^{(l)}$ defined similarly to the first layer. This layered design integrates high-order connectivity, enhancing embeddings for recommendation tasks. Finally, a matrix formulation is used to efficiently update all user and item embeddings:

$$\mathbf{E}^{(l)} = \text{LeakyReLU}\left((\mathbf{L}+\mathbf{I})\mathbf{E}^{(l-1)}\mathbf{W}_1 + \mathbf{L}(\mathbf{E}^{(l-1)} \odot \mathbf{E}^{(l-1)})\mathbf{W}_2\right),$$

where $\mathbf{L}$ is the Laplacian matrix of the user-item interaction graph. After propagating through L layers, we obtain multiple representations for user u and item i, denoted as $\{\mathbf{e}_u^{(1)}, \dots, \mathbf{e}_u^{(L)}\}$ and $\{\mathbf{e}_i^{(1)}, \dots, \mathbf{e}_i^{(L)}\}$, respectivley. The final embeddings for the user and item are formed by concatenating the representations across all layers:

$$\mathbf{e}_u^* = \mathbf{e}_u^{(0)} \parallel \cdots \parallel \mathbf{e}_u^{(L)}, \quad \mathbf{e}_i^* = \mathbf{e}_i^{(0)} \parallel \cdots \parallel \mathbf{e}_i^{(L)},$$

where $\parallel$ represents the concatenation operation. This method enriches the initial embeddings while allowing control over the propagation range through L. Finally, the model estimates the user's preference for the target item using the inner product of the final embeddings:

$$\hat{y}_{\text{NGCF}}(u,i) = (\mathbf{e}_u^*)^\top \mathbf{e}_i^*.$$

While NGCF effectively models high-order connectivity and user-item relationships through embedding propagation, it primarily focuses on static graph structures. It assumes that user preferences and item features remain constant over time, which limits its ability to capture evolving preferences.

By building on the graph propagation mechanisms introduced in NGCF, Dynamic Graph Neural Networks for Sequential Recommendation (DGSR) extends the framework to capture temporal patterns and adapt to dynamic user-item interactions, offering a more robust solution for sequential recommendation tasks. The following sections delve into the DGSR framework, detailing its dynamic graph construction, temporal modeling, and optimization techniques.

2.3.3 Dynamic Graph Neural Networks for Sequential Recommendation

Motivated by NGCF, Dynamic Graph Neural Networks for Sequential Recommendation (DGSR) integrates temporal dynamics with graph-based modeling. While various methods of graph neural networks have demonstrated impressive results, they often fall short of explicitly capturing the dynamic, collaborative signals between different user sequences. Most models focus on each user's isolated sequence rather than considering the higher-order connections between users. (ZHANG et al., 2022)

Unlike traditional methods that isolate user sequences, DGSR integrates user interactions within a dynamic graph structure. This graph-centric perspective enables the model to capture sequential information and collaborative signals among user sequences. The DGSR framework consists of four primary components: dynamic graph construction, sub-graph sampling, Dynamic Graph Recommendation Networks (DGRNs), and a prediction layer. Each component enables the model to learn dynamic user preferences effectively. Figure 2.8 shows the framework of the model.

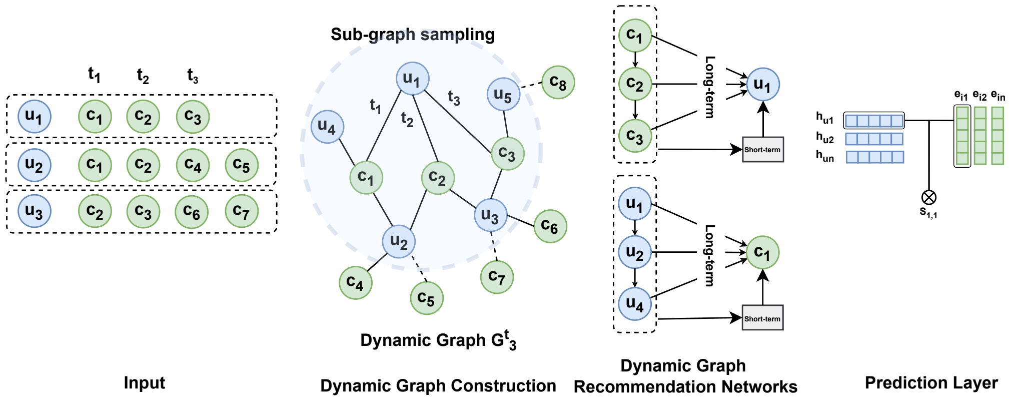

Input
Sub-graph sampling
Dynamic Graph G^{t}_{3}
Dynamic Graph Construction
Long-term
Short-term
Dynamic Graph Recommendation Networks
Prediction Layer

Figure 2.8: The DGSR framework uses input sequences to construct a dynamic graph. Sub-graphs are constructed for each user and item node to aggregate messages and update the node embedding. Each sub-graph is limited to a maximum sequence length. Long-term and short-term interests are modeled using an order-aware attention mechanism. Final user embeddings from each layer are concatenated to form the user's unified embedding. Prediction is then made by matrix multiplication between the users' unified and item embeddings (ZHANG et al., 2022).

DGSR begins by transforming all user-item interaction sequences into a dynamic graph. This graph represents users and items as nodes, while edges denote their interactions. Each edge is weighted with a timestamp. For instance, the timestamp indicates when the interaction occurred and the position of the interaction within the sequence.

2.3.3.1 Sub-graph sampling

DGSR uses sub-graph sampling to process dynamic graphs efficiently. Starting from a user node, it retrieves recent interactions and expands to multi-hop neighbors to order m. This approach captures relevant direct and indirect interactions while reducing noise and computational complexity. (ZHANG et al., 2022)

Moreover, data split also takes place during sub-graph sampling. In contrast to the known split methods, which split the data into training and testing sequences, DGSR leaves out the last interaction within each interaction sequence for testing, the second last for validation, and the rest of the sequence for training, as illustrated in figure 2.9. The model then predicts the link between the target user and the last interaction.

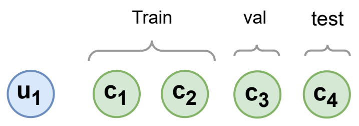


Figure 2.9: Splitting interactions into training, validation, and testing (ZHANG et al., 2022).

2.3.3.2 Message Propagation and node updates

DGSR incorporates two key mechanisms: message propagation and node updating. These mechanisms work together to iteratively refine node representations. The message propagation mechanism accounts for the sequential order of interactions and the relationships between users and items. This is achieved by considering two types of information for user and item nodes: long-term preferences/characteristics and short-term preferences/characteristics.

Long-term preferences of a user node u are derived from all historical interactions, reflecting their inherent characteristics and general preferences. Similarly, the long-term characteristics of an item node i reflect its general properties based on interactions with multiple users. To encode these long-term attributes, DGRN employs an order-aware dynamic attention mechanism that adaptively weights neighbor contributions:

$$e_{ui} = \frac{\left(W_2^{(l-1)} h_u^{(l-1)}\right)^T \left(W_1^{(l-1)} h_i^{(l-1)} + p_{riu}^K\right)}{\sqrt{d}}, \alpha_{ui} = \text{softmax}(e_{ui}),$$

where r_{iu} is the relative order of item i in u's sequence, p_{riu}^K is the relative-order embedding, and d is the embedding dimension. The long-term preference is then aggregated as:

$$h_u^L = \sum_{i \in N_u} \alpha_{ui} \left(W_1^{(l-1)} h_i^{(l-1)} + p_{riu}^V\right).$$

Short-term preferences capture recent interests and trends. These are computed using attention mechanisms that emphasize the last interaction relative to historical interactions:

$$h_u^S = \sum_{i \in N_u} \alpha_{ui} h_i^{(l-1)},$$

$$\alpha_{ui} = \text{softmax}\left(\frac{\left(W_3^{(l-1)} h_{i_{N_u}}^{(l-1)}\right)^T \left(W_2^{(l-1)} h_i^{(l-1)}\right)}{\sqrt{d}}\right),$$

$$h_i^S = \sum_{u \in N_i} \beta_{iu} h_u^{(l-1)},$$

$$\beta_{iu} = \text{softmax}\left(\frac{\left(W_4^{(l-1)} h_{u_{N_i}}^{(l-1)}\right)^T \left(W_1^{(l-1)} h_u^{(l-1)}\right)}{\sqrt{d}}\right),$$

where parameters $\mathbf{W}_3$ and $\mathbf{W}_4 \in \mathbb{R}^d$ are to control the importance of the last interaction. Once the message propagation mechanism aggregates long-term and short-term information, the node representations are updated for the next layer. The user and item node embeddings are updated using the following equations:

$$h_u^{(l)} = \tanh\left(W_3^{(l)} \left[h_u^L \| h_u^S \| h_u^{(l-1)}\right]\right),$$

where $W_3^{(l)}$ is a trainable weight matrix, and $\|$ represents the concatenation operator.

$$h_i^{(l)} = \tanh\left(W_4^{(l)}\left[h_i^L \| h_i^S \| h_i^{(l-1)}\right]\right).$$

2.3.3.3 Prediction layer

The prediction layer in DGSR finalizes the task of sequential recommendation by predicting the next item a user is likely to interact with. It transforms the next-item prediction into a link prediction task. After passing the sub-graph through L layers of DGRN, each user node u obtains embeddings from all layers, denoted as:

$$\{h_u^{(0)}, h_u^{(1)}, \ldots, h_u^{(L)}\}$$

Each embedding $h_u^{(l)}$ captures different aspects of the user's preferences (X. WANG et al., 2019a). To form a comprehensive representation of the user throughout all layers, these embeddings are concatenated into a single vector:

$$h_u = h_u^{(0)} \| h_u^{(1)} \| \ldots \| h_u^{(L)},$$

where $||$ represents the concatenation operator. This aggregated embedding h_u encodes high-order collaborative information and dynamic sequential preferences. Given the aggregated user embedding h_u and candidate item embeddings e_i, the model computes the likelihood of a link between u and i by multiplying both vectors to get the similarity scores between each pair:

$$s_{ui} = h_u^T e_i$$

The vector of scores for all candidate items is:

$$s_u = \{s_{u1}, s_{u2}, .., s_{u|I|}\}$$

The model predicts the next item i^* by selecting the item with the highest score:

$$i^* = \arg\max_{i \in I} s_{ui}.$$

A cross-entropy loss function is applied to optimize the model. For each user u, the scores are normalized using the softmax function:

$$\hat{\mathbf{y}}_u = \text{softmax}(\mathbf{s}_u),$$

where $\hat{\mathbf{y}}_u$ is the probability distribution over the candidate items. The loss function is then defined as:

$$\mathcal{L} = -\sum_u \sum_{i \in I} y_{ui} \log \hat{y}_{ui} + (1 - y_{ui}) \log(1 - \hat{y}_{ui}) + \lambda \|\Theta\|^2,$$

where y_{ui} is the ground-truth label, equal to 1 if user u interacted with item i, and 0 otherwise. The term $\hat{y}_{ui}$ represents the predicted probability for item i, while Θ denotes all trainable parameters of the model. Finally, λ is a regularization coefficient used to prevent overfitting by controlling the complexity of the model.

One disadvantage of this approach and similar approaches such as NGCF (X. Wang et al., 2019b) and SelfGNN (Y. Liu et al., 2024) is that the graph training and prediction occur in a transductive setting. As previously stated, the model attempts to predict absent information within an existing graph, such as link predictions or node classification. The trained DGSR cannot generalize to unseen graphs or nodes. In practical environments, users continue to join the network to interact with items. In scenarios where user or item features are unavailable, the model cannot generate reliable embeddings for real-time recommendations. This issue will be addressed in detail in our methodology.

Chapter 3

Related Work

3.1 Sequential recommendation

Sequential recommendation has evolved significantly with the advent of deep learning approaches. This section provides a comprehensive review of the major methodological developments in this field. Early approaches to sequential recommendation leveraged RNNs to capture temporal dynamics. GRU4Rec (HIDASI et al., 2016) pioneered this direction by using Gated Recurrent Units for session-based recommendation, introducing specialized ranking losses for recommendation tasks. NARM (J. LI et al., 2017) combined attention mechanisms with RNNs to capture users' main purposes in the current session, introducing a hybrid encoder with global and local preferences. STAMP (Q. LIU et al., 2018) proposed a short-term attention priority model that captures users' general and current interests through an attention-based network. Time-LSTM (ZHU et al., 2017) extended the LSTM architecture to model time intervals between consecutive interactions, introducing time gates to capture temporal aspects. Hierarchical RNN (J. WU et al., 2020) proposed an RNN model that captures session-level and user-level sequential patterns, enabling personalization in the session-based recommendation.

The success of attention mechanisms in natural language processing has inspired numerous sequential recommendation approaches. CSAN (HUANG et al., 2018) addressed the limitations of RNN-based sequential recommendation by using feature-wise self-attention to capture heterogeneous and polysemous user behaviors in a common latent semantic space. The model incorporates forward and backward position encoding matrices to model dynamic contextual dependencies. SASRec (KANG & MCAULEY, 2018) pioneered self-attention in the sequential recommendation, demonstrating superior performance in capturing long-term dependencies with relatively shorter sequences than RNN. BERT4Rec introduced (SUN et al., 2019) a bidirectional transformer model for sequential recommendation systems that improves upon traditional unidirectional approaches by using a Cloze task to predict masked items based on both left and right context.

TiSASRec (J. Li et al., 2020) improved upon traditional approaches by explicitly incorporating both position and time interval information between user interactions using self-attention mechanisms. The model demonstrated superior performance across sparse and dense datasets, showing the value of considering temporal information rather than just interaction order. SSE-PT (L. Wu et al., 2020) improved upon SASRec by incorporating user embeddings and Stochastic Shared Embeddings regularization, achieving better interpretability and the ability to handle long sequences. DuoRec (Qiu et al., 2022) addressed the representation degeneration problem in sequential recommender systems by introducing a contrastive regularization method that reshapes sequence representation distribution using model-level augmentation based on Dropout.

Graph-based approaches have emerged as a powerful tool for learning complex user-item interactions. NGCF (X. Wang et al., 2019a) leverages the connections between users and items, forming a bipartite graph. Instead of treating user and item embeddings as independent entities, NGCF explicitly incorporates their interaction patterns into the embedding learning process to capture the underlying collaborative signals in the graph structure. SR-GNN (S. Wu et al., 2019) pioneered the use of gated GNNs for session-based recommendation by modeling sessions as graphs, capturing item transitions through the interaction sequence.

GCE-GNN (Z. Wang et al., 2020) enhances session-based recommendation by combining global-level item representations learned through session-aware attention on a global graph with session-level representations learned via GNN on the current session graph using an attention mechanism. DGTN (Zheng et al., 2020) introduced a dual graph transformer network that models item transitions and attribute correlations. MA-GNN (Ma et al., 2019) utilizes a shared memory module that captures short-term and long-term dependencies.

3.2 BIM command prediction

A BIM modeler's sequence contains the logical modeling operations to complete a model. To increase modeling efficiency, an approach for predicting sequential design commands has been developed by (PAN & ZHANG, 2020b) using an RNN, offering an opportunity to uncover hidden insights within BIM event logs. To comprehensively analyze a series of design commands within these logs, an RNN with sequential memory is constructed to learn features from the extracted sequential data and predict the next possible design commands. This RNN-based prediction method was tested on a real dataset from a BIM event log, specifically the "Create" journal file, which contains 57,915 command records. The task was treated as a multi-classification problem, with hundreds of design commands grouped into six categories. The RNN achieved an overall accuracy of 63.86%. This research was innovative in advancing the state of knowledge by utilizing RNNs for event log data mining to accurately predict design command sequences and enhancing the state of practice by offering an intelligent modeling solution that can help designers improve the efficiency and quality of the design process (PAN & ZHANG, 2020b). On the other hand, the prediction was limited to the command class level and didn't predict the next command itself.

(PAN & ZHANG, 2020a) used event log data stored in Autodesk Revit journal files to extract the modeller-command interaction sequence. Their dataset included 2647 projects with 853,520 command lines containing 31,040 command types. After cleaning, the dataset contained 377 projects with 352,056 command lines and 289 command types. The extracted commands were then classified into 14 command classes based on their function: Create Dimensions, Objects, View, Element, Delete Element, Furniture, Elevator, etc., or Other. The next possible design command class was then predicted using a LSTM neural netowrk, achieving a prediction accuracy of 70.5% on the test set. Although this approach produced slightly higher prediction accuracy than the previous, it is also limited to predicting command classes rather than a specific possible command.

(GAO et al., 2021) proposed a new data structure of command-object graph derived from 3D modeling event log. This data structure highlights the cause-and-effect relation between the executed command and the modeled object to better describe the 3D modeling process. Event log files were collected from student project work on a facade model. In a later research, (GAO et al., 2022) derived command sequences from this data structure and used them as input to a standard transformer model to predict the next most probable command. This model achieved 94% on top one command prediction. The augmentation based on their proposed command-object graph data structure improved the prediction accuracy by up to 1.75 times. However, this data structure is not publicly accessible by any software user and requires manual updates (JANG et al., 2023).

3.3 Research gap

The following research gaps were identified in the literature:

- Current neural network approaches for BIM command prediction are limited to predicting command categories rather than specific commands, reducing their practical usage.
- While GNN-based sequential recommendation has proven effective for e-commerce, its application to BIM command prediction remains unexplored. BIM event logs differ from e-commerce interactions in that they have longer interaction sequences, which can exceed 2000 interactions.
- Current GNN-based sequential recommendation methods encounter scalability challenges in production environments such as BIM authoring tools. The transductive nature of these methods restricts their ability to generalize to new users and evolving graphs, thereby hindering real-time predictions.

Chapter 4

Methodology

Our method is based on the DGSR architecture that was originally designed for sequential recommendation in commercial applications to predict the next command of a BIM modeler. The data is extracted from raw event logs of the BIM authoring tool Vectorworks from Nemetschek and preprocessed to obtain the meaningful modeling sequences of the modelers. As a result of our work, a trained DGSR model can be utilized to get real-time predictions without continuously retraining the whole graph network.

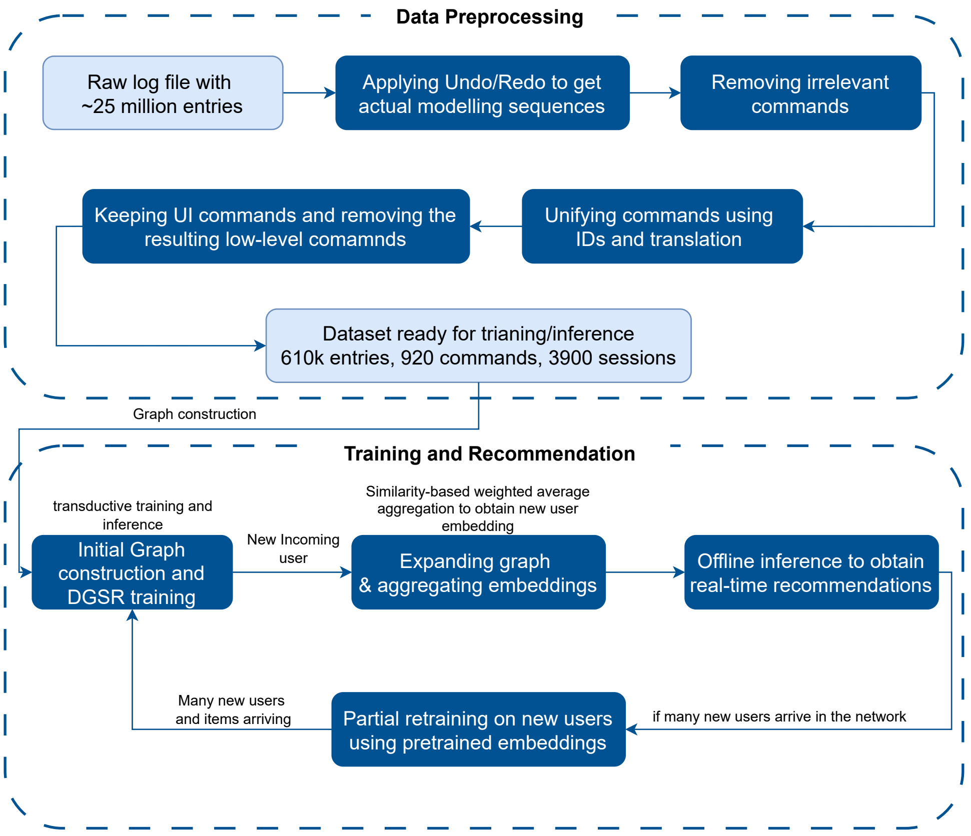

Data Preprocessing
Raw log file with ~25 million entries
Applying Undo/Redo to get actual modelling sequences
Removing irrelevant commands
Keeping UI commands and removing the resulting low-level comamnds
Unifying commands using IDs and translation
Dataset ready for trianing/inference 610k entries, 920 commands, 3900 sessions
Graph construction
Training and Recommendation
transductive training and inference
Initial Graph construction and DGSR training
New Incoming user
Similarity-based weighted average aggregation to obtain new user embedding
Expanding graph & aggregating embeddings
Offline inference to obtain real-time recommendations
Many new users and items arriving
Partial retraining on new users using pretrained embeddings
if many new users arrive in the network

Figure 4.1: illustrates the workflow from filtering the raw event log files until generating the next command prediction.

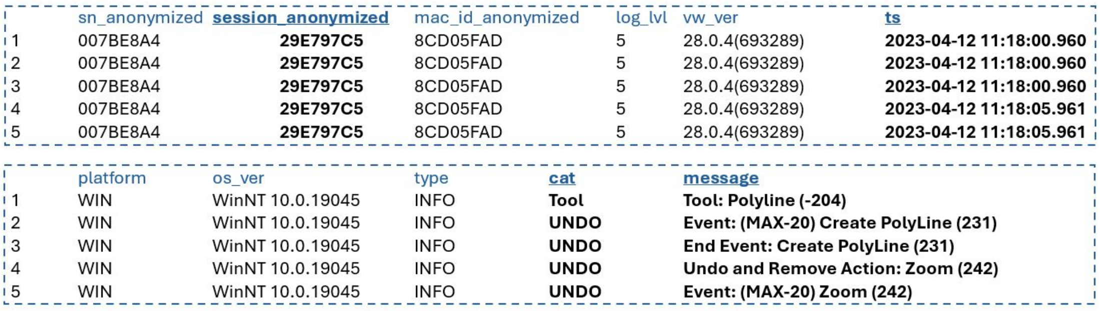
Figure 4.1 illustrates our pipeline to obtain a real-time BIM command prediction tool. The method starts with a raw BIM event log file automatically generated during modeling using a Vectorworks authoring tool in a "comma-separated value" file. It serves as the input to our pipeline. The exact contents of the raw log file will be presented in the upcoming sections.

Subsequently, the CSV file is preprocessed to extract logical and reliable modeling steps. The refined data is then employed in constructing a dynamic heterogeneous graph comprising user and command nodes, with edges between them representing interactions and weighted with timestamps to reflect the order of interactions. Subsequently, the DGSR is trained on the graph network to obtain the final user and command embeddings. If all users were present during the training phase, the model could generate predictions as the modeling sequence expands as a link prediction between the user and command nodes. However, when a new user is introduced, the model cannot create predictions for them without retraining. This marks the culmination of our method, where the graph is expanded to encompass the new user node, and the edges between it and the commands with which it interacts are constructed. A similarity-based aggregation of the old user embeddings is then carried out to obtain a reliable embedding upon which a prediction can be made.

In the upcoming sections, each step of our workflow will be thoroughly explained, showing all inputs and outputs utilized. A systematic analysis will be provided, showing the process of preprocessing the data, passing by training, and ending with prediction.

4.1 Dataset analysis

The dataset employed in this thesis is the raw event log files of Vectorworks, which were collated from 1,000 users over the course of a day using Vectorworks. They comprise 25,397,138 entries recorded in at least 8 languages. The users' data were anonymized before the dataset was delivered to comply with the requisite privacy and security standards. Each user's modeling commands are divided into sessions with a total of 1556 sessions. A session is recorded once a user launches the program until closing it. If a user reopens the program again, his modeling actions are recorded under a new session ID in the "session_anonymized" column. The timestamp at which the command is executed is recorded in column ts. The command itself is recorded in the column message.



	sn_anonymized	session_anonymized	mac_id_anonymized	log_lvl	vw_ver	ts
1	007BE8A4	**29E797C5**	8CD05FAD	5	28.0.4(693289)	**2023-04-12 11:18:00.960**
2	007BE8A4	**29E797C5**	8CD05FAD	5	28.0.4(693289)	**2023-04-12 11:18:00.960**
3	007BE8A4	**29E797C5**	8CD05FAD	5	28.0.4(693289)	**2023-04-12 11:18:00.960**
4	007BE8A4	**29E797C5**	8CD05FAD	5	28.0.4(693289)	**2023-04-12 11:18:05.961**
5	007BE8A4	**29E797C5**	8CD05FAD	5	28.0.4(693289)	**2023-04-12 11:18:05.961**

	platform	os_ver	type	cat	message
1	WIN	WinNT 10.0.19045	INFO	**Tool**	**Tool: Polyline (-204)**
2	WIN	WinNT 10.0.19045	INFO	**UNDO**	**Event: (MAX-20) Create PolyLine (231)**
3	WIN	WinNT 10.0.19045	INFO	**UNDO**	**End Event: Create PolyLine (231)**
4	WIN	WinNT 10.0.19045	INFO	**UNDO**	**Undo and Remove Action: Zoom (242)**
5	WIN	WinNT 10.0.19045	INFO	**UNDO**	**Event: (MAX-20) Zoom (242)**

Figure 4.2: A sample of Vectorwork's raw event log file.

As illustrated in figure 4.2, commands are classified into three principal categories: Tool, Menu, and UNDO. Tool entries are buttons selected from the Tool menu within the user interface. Similarly, Menu entries are defined as buttons selected from the "Menu" menu within the user interface. In contrast, UNDO entries identify commands that are undoable or redoable. UNDO commands are further divided into several categories. For instance, Event, End Event, Undo Event, Redo Event, Undo and Remove Action, Destroy Event, Event Name Change, Begin Internal Event, Abort Event.

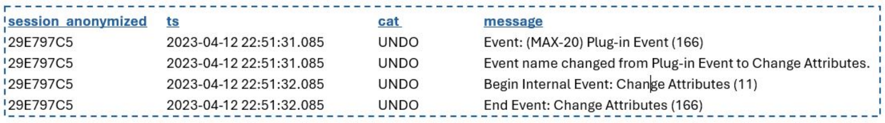
Table 4.1: Dataset Statistics

Metric	**Count**
Rows	25,397,138
Users	1,000
Sessions	1,556
Unique messages	9,541
Languages	7
English entries	15,106,675
Menu	136,157
Tool	964,911
UNDO	24,296,070
End Event(UNDO)	5,833,158

To gain insight into the generation of the raw event log file and to understand what each message denotes, a series of modeling steps were conducted, emulating the commands recorded in the event log. For each button selected from the user interface, the corresponding button is recorded in the log file, categorized as 'Tool' or 'Menu'. Such commands may also be designated as high-level insofar as they do not usually elicit a direct effect on the modeled object. Such records indicate the transition from the neutral mode to the selected tool or menu command. Upon the utilization of a command, a UNDO command is recorded in the log file, thereby marking the commencement of a modeling event. Upon the user's completion of the aforementioned command, a subsequent UNDO entry is recorded in the log file, signifying the successful execution of the modeling event.

To illustrate, users who wish to create a polyline will select the "Create Polyline" option from the "Tool" menu, entering the line drawing mode. The user's actions are recorded in the log file as "Tool: Create Polyline (231)" immediately. Upon clicking on a point to commence line drawing, an event "Event: Create Polyline (231)" is created, designating the start of the modeling action. Subsequently, upon clicking on the desired point where the line ends, "End Event: Create Polyline (231)" is recorded, marking the successful completion of the line drawing. While most Event/End Event entries are initiated via a high-level Tool/Menu button, a few commands are triggered by a keyboard press. For instance, the End Event: Nudge (5) command is triggered by pressing the arrows to move an object slightly.

Furthermore, we found out that "Undo Event" and "Undo and Remove Action" are logged when an event is undone, for example, "Undo Event: Create Polyline (231)". Consequently, "Redo Event" is logged when an event is redone as follows: "Redo Event: Create Polyline (231)". In addition, the entry "Event Name Change" marks the change of a process name to another. This can be clearly seen when events are executed using plug-in tools installed into Vectorworks, which usually occur together with internal events, as shown in figure 4.3.



session_anonymized	ts	cat	message
29E797C5	2023-04-12 22:51:31.085	UNDO	Event: (MAX-20) Plug-in Event (166)
29E797C5	2023-04-12 22:51:31.085	UNDO	Event name changed from Plug-in Event to Change Attributes.
29E797C5	2023-04-12 22:51:32.085	UNDO	Begin Internal Event: Change Attributes (11)
29E797C5	2023-04-12 22:51:32.085	UNDO	End Event: Change Attributes (166)

Figure 4.3: A sample showing the logging of Plug-in actions and the associated internal events and event name changes.

Moreover, the entry "Abort Event" is employed to signify the termination of the insertion mode of any given action. To illustrate, if a user were contemplating the creation of a chained dimension line, they would select the corresponding button in the Tool menu. This action would result in recording both a "Tool" and an "Event" entry. However, prior to the placement of the dimension line, the user may opt to cancel the command, which is typically achieved by pressing the ESC button. In such instances, the "Abort Event" is duly recorded in the log file. Should the user wish to resume the creation of the chained dimension line, an "Event" entry will be recorded upon returning to the insertion mode, as well as an "End Event" upon successfully placing the dimension line. It can, therefore, be considered safe to discard the "Abort Event" and the prior recorded "Event" entry, as they are irrelevant to the modeled objects.

A further challenge was the necessity to accommodate entries in multiple languages. As the dataset was collected from users across the globe, the commands are logged according to the language of the interface in use. Consequently, the same command is logged multiple times under disparate records, despite originating from a single action. Table 4.2 illustrates the manner in which the same command is recorded on multiple occasions due to the use of different languages. It should be noted that all entries have the same localization ID, which will facilitate the unification of the dataset, ensuring that all commands are assigned the same language.

Table 4.2: A sample of different records of the same command due to different languages with the same localization ID.

message
End Event: Polylinie anlegen (231)
End Event: Create PolyLine (231)
End Event: Creëer polylijn (231)
End Event: Crea Polilinea (231)
End Event: Créer une polyligne (231)
End Event: Utwórz polilinię (231)

However, this is not the case with recorded commands triggered when using plug-ins installed in Vectorworks. Those entries share the same localization ID but denote a totally different action, as illustrated in Table 4.3. Furthermore, there are additional commands that are similar in function but different in language. Usually, these could have been easily unified using their localization ID. However, due to language differences, an additional translation step would be required to unify all the commands, which will be discussed in the upcoming section.

Table 4.3: The column on the left shows commands resulting from plug-ins; on the right are commands triggered by other buttons. Both share the same localization ID but have different languages and purposes.

Plug-in messages	**Others**
End Event: 2D Align / Distribute (166)	End Event: Create Objects from Shapes... (-1)
End Event: 2D Ausrichten (166)	End Event: Utwórz obiekty z kształtów... (-1)
End Event: Lijn uit/Verdeel (166)	End Event: Creëer objecten d.m.v. meetkundige vorm... (-1)
End Event: Create Roof (166)	End Event: Line into Segments... (-1)
End Event: Creëer dak (166)	End Event: Segmenter la ligne... (-1)
End Event: Utwórz dach (166)	End Event: Change Plant Grouping (-1)
End Event: Create Line (166)	End Event: Grupuj rośliny/Rozdziel grupę roślin (-1)
End Event: Créer une ligne (166)	End Event: Send to Surface (-1)
End Event: Linie anlegen (166)	End Event: Colloca sulla superficie (-1)

4.2 Data preprocessing

After analysis, we found that the data cannot be used directly for training because it contains many redundancies due to undone commands, irrelevant commands, and duplications due to different languages. Data had to be filtered and preprocessed to extract the actual and most relevant modeling commands. For instance, if we assume that using Tool, Menu, and End Event entries directly from the raw event log is the actual modeling sequence, this will result in strong noise. Undone commands actually result from the natural trial-and-error behavior of a BIM modeler. That's why it was necessary to eliminate the Undo and Redo entries and apply their effect on the dataset. This has been carried out in two steps. First, Redo entries are processed using an algorithm that detects Redo events and then goes up in the dataset until it finds the corresponding Undo entry. Both entries are then removed. Second, the algorithm detects the remaining effective Undo events, which were not Redone, and then searches the dataset upwards to find the corresponding End Event and high-level Tool/Menu command and remove them all. In this way, only the effective buttons and end events are left in the dataset.

Subsequently, the dataset was subjected to a process of elimination, whereby commands that were deemed irrelevant were identified and removed. To illustrate, the frequency of each distinct command within the unprocessed event log was calculated, as demonstrated in figure 4.4. The top 30 frequencies indicate that the most frequently recorded commands are 'zoom', 'changing views', and 'pan'. These observations are logical, given that each modeler will typically zoom in and out and change views on numerous occasions within a relatively short timeframe. However, these commands are irrelevant since they do not contribute to the modeling process. That's why such entries had to be removed from the dataset to avoid imbalance and avoid recommending irrelevant actions. A recommender system could be deployed in another application to recommend the next element to be modeled instead of using a simple command. In this case, changing views or zooming could limit the prediction to a small model area, thereby contributing to a more robust and relevant recommendation. The development of such an application is beyond the scope of this thesis.

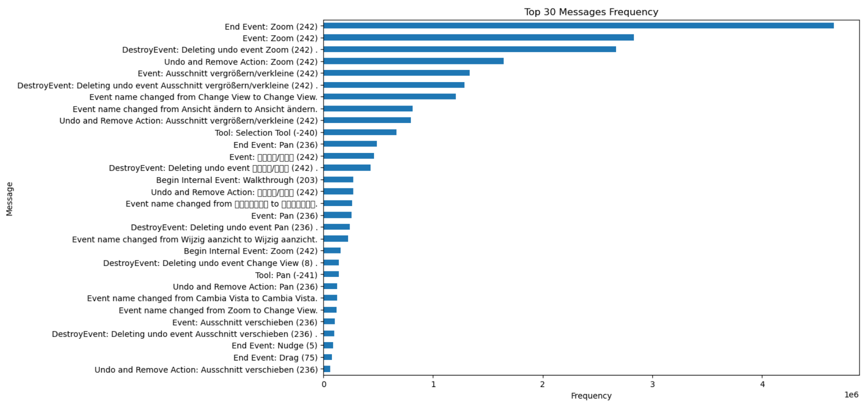
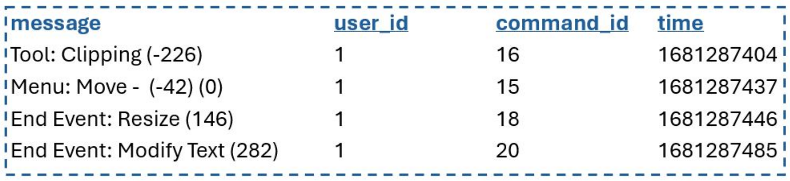


Figure 4.4: Frequency of the top 30 messages in the raw event log file.

Furthermore, low-level End Event entries triggered by high-level Tool or Menu commands are deemed irrelevant. This is because both are the result of a single action. Therefore, retaining them within the dataset is redundant and will result in this action being accorded twice the importance it would otherwise have. The dataset was thus examined to extract each tool and menu command and its resulting end event entry. This was achieved by extracting the preceding and following entries to a tool or menu entry and examining the triggered end event entry. Entries found to occur in close proximity on numerous occasions were assumed to be directly related, and the low-level entry was discarded, retaining only the high-level entry.

Finally, a unification of the remaining multilingual entries had to take place. First, localization IDs were used to align all End Event entries with the English entry. Some commands were recorded in languages other than English only. Those were aligned to German and sometimes Dutch language. Second, entries triggered by plug-in events with the localization ID (166) and other entries with ID (-1) were unified by using the Googletrans library from Google Translate API to translate non-English entries and to know what they mean and then replace them with the English entry. All translated entries were manually checked to ensure correct translation and alignment. All unifying databases have been saved to unify any newly generated log files for future utilization.



message	user_id	command_id	time
Tool: Clipping (-226)	1	16	1681287404
Menu: Move - (-42) (0)	1	15	1681287437
End Event: Resize (146)	1	18	1681287446
End Event: Modify Text (282)	1	20	1681287485

Figure 4.5: Sample of the CSV file that will be used for training

Eventually, timestamps were transformed into UNIX time format, which calculates the time elapsed since 00:00:00 on 1 January 1970 to be used as input to the model. In addition, each unique command and user ID were one-hot encoded, i.e. giving each of them a unique integer value as illustrated in figure 4.5. All of this data is saved in a CSV file.

4.3 Initial graph construction and training

4.3.1 Graph generation and sub-graph sampling

To train the original DGSR model, a heterogeneous graph must first be constructed to accommodate the user and command nodes available at training time. As explained in section 2.3.3.2, the graph construction process begins with preprocessing the input CSV file to ensure that temporal relationships between interactions are correctly captured. This is achieved by calculating each user's relative order of interactions based on the timestamps, reflecting the sequence in which commands were executed. Similarly, the relative order of interactions for each command across all users provides a global perspective on command execution over time.

The heterogeneous graph is constructed as a BIM knowledge base. As illustrated in figure 4.6, The nodes in the graph represent users and commands, while the two-way edges capture the interactions between them. Specifically, the edge type ('command', 'by', 'user') denotes that a user issued a specific command, while ('user', 'pby', 'command') represents the reverse relationship. Timestamps of interactions are stored as edge features for both edge types, enabling the graph to encode temporal dynamics. User and item identifiers (user_id and command_id) are stored as node features, ensuring each node is uniquely identifiable. The graph is implemented using the Deep Graph Library (DGL), and the relationships are defined through a dictionary mapping node pairs to their corresponding interaction data. By combining sequential, temporal, and structural information, this graph effectively represents the relation of BIM modeler interactions with commands, enabling advanced downstream sequential recommendation.

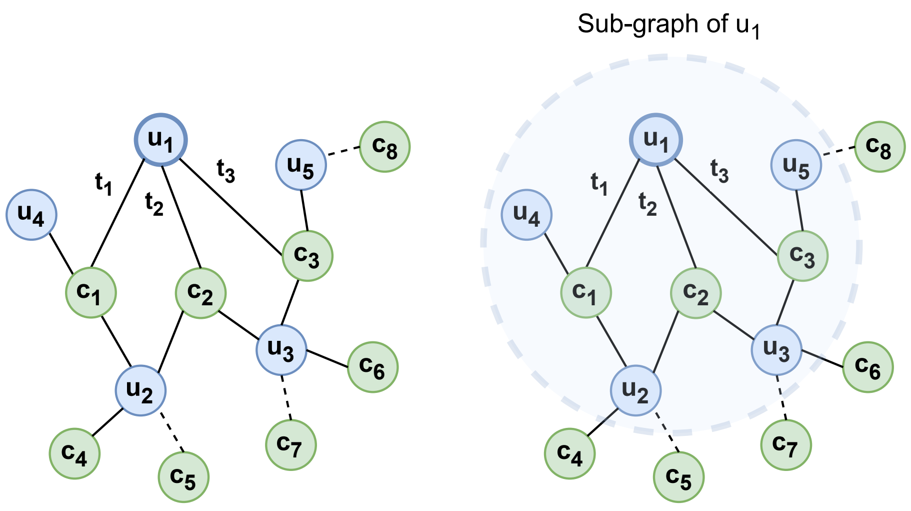


Figure 4.6: An illustration of the 2-hop sub-graph sampling analog to the utilized in (ZHANG et al., 2022).

Although the graph contains all user sequences, training has not started yet. Feeding the whole graph is computationally expensive and would require huge computational power from one side. That's why sub-graphs must be sampled beforehand to accelerate the training process. As illustrated in section 2.3.3, the sub-graph is limited to only two-hop nodes. This means that when a user node is taken as an anchor, all items within the interaction sequence and the other users who interacted with them are sampled, as illustrated in the figure. This is also applied to command nodes.

The resulting computation graphs illustrated in figure 4.7 show the message passing and aggregation flow. For instance, the flow of embeddings from commands to users reflects a user+'s preference. Analog to e-commerce profiles, the aggregated embeddings also reflect a user's profile. The modelers' preferences are narrowed to a smaller pool of commands with which they interact. Similarly, on a command node level, the flow of information from user nodes to command nodes reflects commands' characters (ZHANG et al., 2022).

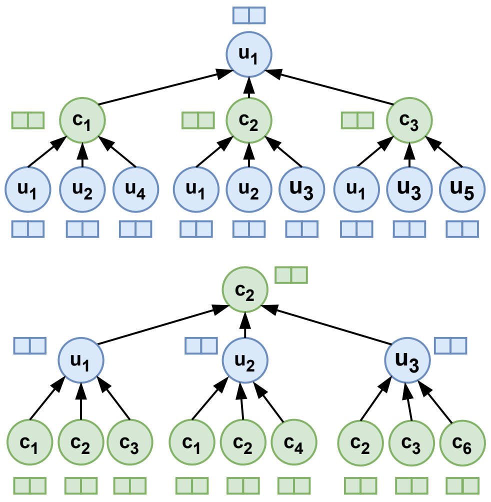


Figure 4.7: Computation graphs of sample user and command nodes after sub-graph sampling. In user graph, first hop nodes are interacted commands and second hop nodes are users who interacted with the same commands as the anchor user.

In order to successfully train and test the model, we employed the same splitting method employed by (X. WANG et al., 2019a), (ZHANG et al., 2022), and (Y. LIU et al., 2024), and illustrated in section 2.3.3.1 so that the last interaction is left for testing, the second last for validation, and the rest for training. The same hyperparameters illustrated in (ZHANG et al., 2022) were utilized during the training of the model on the initial graph.

4.4 Graph expansion and offline inference

After successfully training the DGSR model following the approach proposed by its authors, the method was further developed to simulate real-world production environments. In a typical production scenario, a new modeler joining the team will have no prior modeling history. Since DGSR operates in a transductive manner, where all user nodes must be present during training to learn their preferences, the model cannot predict the next command for a new user who was unseen during training. This limitation arises because, at the model initialization phase, user and command node embeddings are randomly initialized based on their unique IDs assigned during preprocessing. These initial embeddings lack meaningful relationships or connections between them. When such randomly initialized embeddings are passed to the model, it can lead to illogical or unreliable predictions for new users.

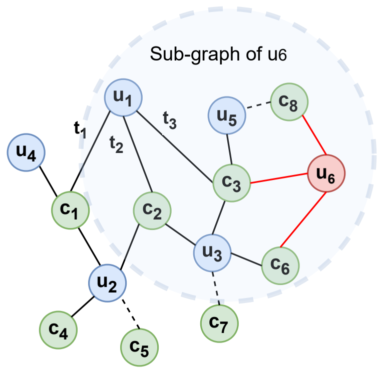


Figure 4.8: The new user node u_6 arrived in the graph and its 2-hop subgraph. The sub-graph shows the users whom u_6 will aggregate from, i.e., users who interact with the same commands.

To address this limitation, we expanded the original graph to include the new user's node and timestamped edges representing their interactions with specific commands. By incorporating these edges, the dynamic development of the new user's interaction sequence was captured. Subgraphs were then constructed for the new user, focusing on their most recent interactions, which simulate the sequential and dynamic nature of user-command interactions within a graph structure. This step ensures that the new user's node can receive the collaborative signals from users with the same preferences within the graph.

To construct a reliable profile for the new user, the embeddings of existing users (who were present during training) were leveraged. First, the interaction sequences of the new user were compared with those of the pretrained users using the Jaccard similarity coefficient. The Jaccard similarity J measures the overlap between two sets of commands normalized by their union, providing an intuitive way to quantify behavioral similarity.

$$J(u_{\text{new}}, u_i) = \frac{|S_{\text{new}} \cap S_i|}{|S_{\text{new}} \cup S_i|},$$

where S_{new} is the set of items (commands) interacted with by the new user, S_i is the set of items (commands) interacted with by an existing user u_i, nominator is the number of shared commands between S_{new} and S_i, and the denominator is the total number of unique commands in their union. Once the similarity scores were obtained, the embeddings of the similar pretrained users were aggregated to generate the embedding for the new user. This aggregation was performed as a weighted average, where the weights correspond to the Jaccard similarity scores. Users with higher similarity contributed more to the final embedding, ensuring that the generated embedding accurately reflected the new user's interaction behavior.

$$\mathbf{E}_{\text{new}} = \frac{\sum_{i \in N} J(u_{\text{new}}, u_i) \cdot \mathbf{E}_i}{\sum_{i \in N} J(u_{\text{new}}, u_i)}$$

In situations where no meaningful similarity could be established between the new user and any pretrained user (e.g., due to a lack of sufficient interaction data), a default embedding was assigned to the new user. This default embedding was computed as the mean of all pretrained user embeddings, providing a fallback representation that captures the general distribution of user preferences observed during training.

The newly generated embeddings for the new users were combined with the pretrained user embedding vector, resulting in an updated set of user embeddings. This updated vector was saved back into the model's state dictionary, enabling the DGSR model to include both the original and new users during inference. The model was then reloaded with the updated embeddings and set to evaluation mode, and predictions were created offline, allowing for real-time predictions. Algorithm 1 shows the summary of our method.

Algorithm 1 The DGSR Framework with New User Integration

Require: $S^u = (i_1, i_2, \cdots, i_k)$, timestamp sequence $T^u = (t_1, t_2, \cdots, t_k)$, all sequences of users, DGRN layer number L, new user u_{new}, initial interactions S_{new} with timestamped edges T_{new}.

Ensure: The next item i_{k+1} of S^u.

```
 1: // Dynamic Graph Construction
 2: Convert all user sequences into a dynamic graph $G$.
 3: if $u_{\text{new}}$ is present then
 4:     Expand $G$ to include $u_{\text{new}}$ and edges $(u_{\text{new}}, i), \forall i \in S_{\text{new}}$.
 5:     Construct subgraph $G^m_{u_{\text{new}}}(t_k)$ to reflect $u_{\text{new}}$'s interactions.
 6:     Measure similarity $J(u_{\text{new}}, u_i)$ for all $u_i \in G$.
 7:     Aggregate embeddings for $u_{\text{new}}$.
 8: else
 9:     Use the subgraph of $G$ for $S^u$.
10: end if
11: Run Algorithm 1 to generate $G^m_u(t_k)$ from $G^{t_k}$.
12: // Initialization of Node Representation
13: $h^{(0)}_u \leftarrow e_u$, $h^{(0)}_i \leftarrow e_i$, $\forall u, i \in G^m_u(t_k)$.
14: // Update of User and Item by DGRN
15: for $l \in [1 : L]$ do
16:     $h^{(l)}_u, h^{(l)}_i \leftarrow$ DGRN$(h^{(l-1)}_u, h^{(l-1)}_i, G^m_u(t_k))$.
17:     $h^{(L)}_u, h^{(L)}_i \leftarrow$ Long-term Information Encoding.
18:     $h^{(S)}_u, h^{(S)}_i \leftarrow$ Short-term Information Encoding.
19:     $h^{(l)}_u \leftarrow \tanh\left(W^{(l)}_3 \left[h^{(L)}_u \| h^{(S)}_u \| h^{(l-1)}_u\right]\right)$.
20:     $h^{(l)}_i \leftarrow \tanh\left(W^{(l)}_4 \left[h^{(L)}_i \| h^{(S)}_i \| h^{(l-1)}_i\right]\right)$.
21: end for
22: // Prediction of Next Item
23: $h_u = h^{(0)}_u \| h^{(1)}_u \| \cdots \| h^{(L)}_u$.
24: Next item $\leftarrow \arg\max_{i \in V}(h^\top_u W p e_i)$.
```

4.4.1 Partial retraining on new users

If many new users arrive in the graph with distinctive training sequences, the offline inference method will not yield reliable predictions. Normally in a transductive setting a whole retraining of the whole graph will then be required. We extended our offline inference method to partially retrain the model to learn user representations. For instance, the graph is expanded to include all the new users with their interaction sequences. Then, their sub-graphs are sampled using the same approach previously described.

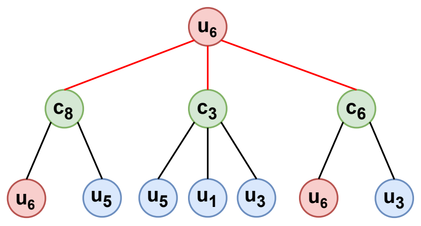


Figure 4.9: Assuming user u_6 one of the many new users arriving at the graph. This figure illustrates their computations graph, showing old users in the 2-hop level enabling collaborative signal transfer.

Instead of passing the sub-graphs of all users, including old and new users, to the model, the subgraphs of only the new users are passed to the model. Since the sub-graphs are of second degree, old user nodes will appear in the 2-hop level, enabling message passing from the old user level to the command level and ending at the new user level. Eventually, the node update will update the new users' representations, reflecting their true preferences as well as the collaborative signals from old users as illustrated in figure 4.9. Consequently, the whole network's accuracy is raised above this of the offline inference method in a shorter time compared to retraining using the whole graph.

Chapter 5

Results and analysis

In this chapter, the model definition and parameters are introduced, as well as the hardware used to apply our methodology. Next, the results of each step of our method will be represented and analyzed.

5.1 Preprocessing results

The dataset preprocessing process aimed to remove redundancies and irrelevant commands in the raw BIM event logs to improve its suitability for training. The key steps included filtering undone commands, removing irrelevant commands, unifying multilingual entries, and final preparation in which Undo and Redo entries were eliminated. The resulting dataset properties are described in table 5.1.

Table 5.1: Properties of the dataset after preprocessing.

Metric	**Count**
Users	1000
Sessions	1,556
Commands	920
Total Interactions	610,768
Average session length	392
Sparsity (sessions)	57.39%
UNDO commands	475,193
Tool commands	90,276
Menu commands	45,299
validation instants	1430
test instants	1430

Firstly, Redo events were identified, and the Redo and corresponding Undo entries were removed. Effective Undo commands (not Redone) were detected, and associated high-level commands and End Events were removed to retain only effective modeling actions. Secondly, Commands such as "zoom," "changing views," and "pan," which are frequently used but do not contribute to modeling, were excluded to avoid imbalance and irrelevance in recommendations. Low-level End Events triggered by high-level Tool/Menu commands were deemed redundant and removed.

The resulting dataset now contains commands directly affecting the modeled object, as described in figure 5.1. Thirdly, localization IDs were used to standardize End Event entries in English. Commands recorded in other languages were manually verified and translated to English using the Google Translate API. Unified databases were created for future use. Finally, timestamps were converted to UNIX time format, and commands and user IDs were assigned unique IDs for modeling. The processed dataset was then saved in a CSV format for training and used as input to the network.

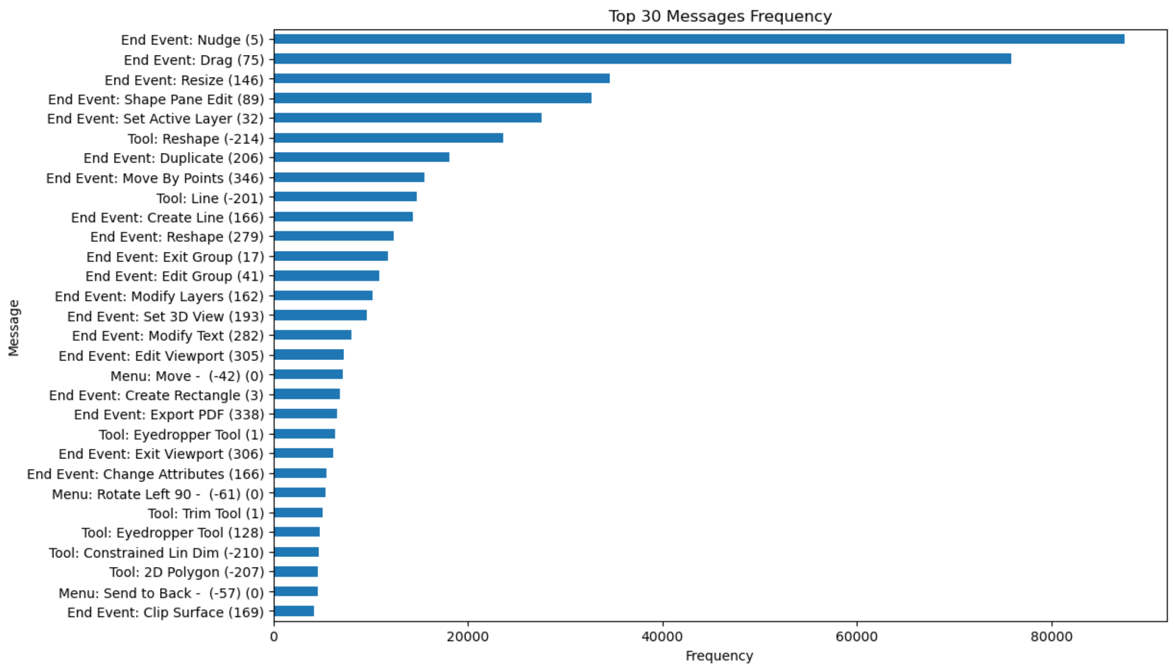


Figure 5.1: Frequency of the top 30 messages in the dataset after preprocessing.

5.2 Model paramters

The proposed method for training as well as partially retraining the model on new users leverages several key packages and tools, including PyTorch for building and training the model as well as tensor operations, DGL for handling graph-based data, and pandas and numpy for data manipulation and numerical operations. The model, DGSR, is initialized with a hidden state size of 50 initialized using PyTorch using user and command IDs, dropout rates of 0.3 for both feature and attention mechanisms, and a layer count of 3 for the GNN architecture. Adam optimizer was configured with a learning rate of 0.001 and weight decay (L2 regularization) of 0.0001. The loss function employed is Cross-Entropy Loss, which evaluates the discrepancy between the predicted and actual labels.

The training process includes batching with a size of 50 and uses multi-threaded DataLoader instances for efficient data handling. This means that each batch contains 50 user sub-graphs. Maximum user and command lengths are initizalied with 50, denoting the maximum length of interactions considered in a sub-graph. A key innovation in this implementation is the incorporation of pretrained user embeddings, updated with embeddings generated for new users based on their similarity to existing users, calculated using Jaccard similarity. This ensures a smooth adaptation of the model to previously unseen user interactions without full retraining.

5.3 Hardware and software

The network employed in this thesis was trained and tested on a laptop with a Windows 10 operating system, a 3.20 GHz AMD Ryzen 7 5800H processor, an 8 GB GeForce RTX 3070 GPU, and 16 GB DDR4-3200 RAM. In addition, another desktop was used to load larger graphs with a Windows 10 operating system, an Intel(R) i5 processor, a 5 GB GPU, and 32 GB 2400 MHz RAM, noting that larger graphs require higher memory resources. DGL version 2.0.0 with CUDA version 12.1 and PyTorch version 2.2.2. Further packages and libraries are mentioned in the DGSR git repository [1]. In addition, due to large memory requirements, mixed precision training was used to decrease memory requirements. Therefore, bigger graphs were used as input for the model.

5.4 Results

In this study, Recall@n and NDCG@n metrics were used to evaluate the utilized network's performance and the developed methodologies. Those metrics are widely used in sequential recommendation models. Recall@N measures the fraction of relevant items successfully retrieved among the top-N recommendations. It is a widely used metric in recommendation systems to evaluate how well a model captures the relevant items in its predictions.

[1] https://github.com/CRIPAC-DIG/DGSR

$$\text{Recall@N} = \frac{\text{Number of relevant items in top-}N}{\text{Total number of relevant items}}$$

Recall@N focuses on whether the system retrieves as many relevant items as possible within the top-N positions. A higher Recall@N indicates better performance covering relevant items for a given user.

Normalized Discounted Cumulative Gain (NDCG) is a metric that evaluates the ranking quality of a recommendation system by considering both the relevance of the recommended items and their positions in the ranked list. Specifically, NDCG@10 focuses on the top 10 recommended items. The calculation of NDCG involves three steps: first, the Discounted Cumulative Gain (DCG) is computed, which uses a logarithmic discount factor to reduce the contribution of relevant items appearing lower in the ranking. The formula for DCG@10 is given as:

$$\text{DCG@10} = \sum_{i=1}^{10} \frac{rel(i)}{\log_2(i+1)}$$

where $rel(i)$ denotes the relevance score of the item at position i. Next, the Ideal DCG (IDCG) is calculated by assuming a perfect ranking of the top 10 items. Finally, NDCG@10 is obtained by normalizing DCG@10 using IDCG@10:

$$\text{NDCG@10} = \frac{\text{DCG@10}}{\text{IDCG@10}}$$

This normalization ensures the metric is bounded between 0 and 1, where a value closer to 1 indicates a higher quality ranking. NDCG@10 is particularly useful because it measures the presence of relevant items in the top 10 and rewards the system for placing them in higher-ranked positions, reflecting the importance of rank in recommendation quality.

5.4.1 Training on initial graph

In this phase, the DGSR model was adopted to learn user and command representation and predict the user's next command based on his interaction sequence. The first step was training the model on the initial graph that simulates our knowledge base. The dataset used as input included the original 1000 users divided into 1556 sessions. Table 5.1 shows the input data properties after preprocessing raw event logs. User sequences with less than 3 interactions were excluded from training because they might induce noise in the network. That's why validation and test instants are less than the number of users, although they were normally supposed to be equal. The model training results are represented in table 5.2.

Table 5.2: Evaluation metrics of next command prediction as a link prediction on 100% of sessions.

Metric	**Count**
Recall@5	0.8497
Recall@10	0.9098
NDCG@5	0.7529
NDCG@10	0.7723

Although this method yields high prediction accuracy, it cannot inherently generalize to unseen users nor generate real-time predictions. The model's architecture is designed for a transductive setting, meaning it cannot be directly transferred to new graphs or learn embeddings for new users on the fly. Consequently, the model must be retrained on the entire graph to utilize the interaction history of all users for generating predictions for a new user. This approach is computationally expensive and impractical due to the long training times involved. For instance, second-degree subgraph sampling for a dataset with 3,900 sessions, approximately 610,000 interactions, and a maximum user and command sequence length of 50 requires about 30 minutes. Additionally, each epoch takes around 2.5 hours to learn representations for all users and commands. With 10 epochs, the total training time is approximately 25 hours.

Despite these limitations, the results achieved with this approach demonstrate higher accuracy compared to those reported by the authors of the original model. This improvement can be attributed to the dense interaction matrix in our dataset, which has a density of 57.39%, compared to 0.01%, 0.04%, and 0.08% in the datasets used to evaluate the original model. Furthermore, our dataset features an average session length of approximately 392, significantly longer than the average session lengths of 7.6, 9.3, and 27.6 in the original datasets. This greater density and session length provide richer interaction data, contributing to improved performance. We didn't compare our evaluation metrics with the authors since the datasets differ totally in properties.

Random sequences where the model failed to recommend accurate results were selected and examined in detail to understand the reasons behind these failures. Upon analysis, it was observed that the label commands in most cases corresponded to commands with very low occurrence in the dataset, making them underrepresented in the training process. This underrepresentation likely led to insufficient learning of embeddings for these rare commands, thereby reducing the model's ability to predict them accurately.

5.4.2 Graph expansion and offline inference

We used another data split approach to test the accuracy of our offline inference method. First, interaction sessions were further broken down into shorter sessions based on timestamps. For instance, if a session has a gap between two consecutive interactions of more than 15 minutes, a cut is made, and a new session starts. This resulted in a more sparse interaction matrix and shorter average session lengths in comparison to the dataset used in the step before. Secondly, those sessions were split into training and testing datasets. This was executed by sampling the minimum number of sessions with all the unique commands in the training split to ensure the model doesn't encounter unseen items in the testing sequences. Sessions were then randomly added to the training split until an 85-15 split was reached. New input dataset properties are listed in table 5.3.

Table 5.3: Properties of the new dataset after session splits.

Metric	**Count**
Users	1000
Sessions	3900
Commands	920
Total Interactions	610,768
Average session length	156
Sparsity (sessions)	82.97%
UNDO commands	475,193
Tool commands	90,276
Menu commands	45,299

A graph was then constructed with only the training dataset, and the sub-graphs were sampled. Then, the model was trained on it to learn user and command representations, and the model state dictionary was saved. Next, a new graph was constructed, including training and testing datasets. Subsequently, embeddings were aggregated from the pre-trained sessions to the new sessions based on similarity in interaction sequences, i.e., from sessions in the training set to sessions in the testing set. The model state dictionary was then updated to include the new users' embeddings and passed to the model in evaluation mode to predict the next command.

Table 5.4: Evaluation metrics of offline inference method on the testing dataset.

Metric	Count
Recall@5	0.7942
Recall@10	0.8537
NDCG@5	0.67
NDCG@10	0.7072

The last interaction in the sequence was made unique to mitigate the risk of the model overfitting to repeated interactions within a session. Specifically, if a user engages with the same command multiple times before the final interaction, the earlier repetitions are excluded from the sequence, as illustrated in Figure 5.2. This approach ensures that the recommendation process leverages collaborative signals drawn from the collective behaviors of users with similar interests rather than being overly influenced by repetitive actions from an individual user. By emphasizing the uniqueness of the final interaction, the model focuses on capturing meaningful patterns that are more likely to generalize across users and scenarios. This method reduces noise in the input data and aligns the training process to predict diverse and relevant recommendations for the user's next action. Furthermore, it enhances the model's ability to identify significant interaction trends, thereby improving the quality and robustness of the sequential recommendations.

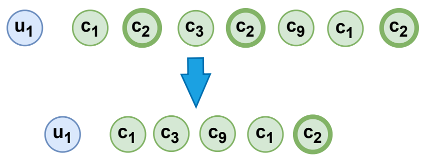


Figure 5.2: Repeated interactions with the same command are removed, retaining only the final unique interaction in the sequence to improve model generalization and prevent overfitting.

These results confirm that the offline inference method provides a practical alternative for generating embeddings for new users without retraining the entire model. Although the accuracy slightly decreases compared to the initial training results, the trade-off between efficiency and accuracy is reasonable as it can generate predictions in a real-time manner.

5.4.3 Graph expansion and partial retraining

If many users arrive in the production environment, the accuracy of the offline inference method will deteriorate due to the distinctive interaction sequences. That's why boosting the representation of the new sessions was necessary to improve the prediction accuracy. This was executed based on our method, which was explained in the previous chapter. The same dataset used in the offline inference method was also used for this method to measure the improvement in accuracy. Table 5.5 illustrates the results after partial retraining.

Table 5.5: Evaluation metrics of partial retraining method on the testing dataset.

Metric	Count
Recall@5	0.8142
Recall@10	0.8761
NDCG@5	0.6975
NDCG@10	0.732

Although the model's accuracy after partial retraining didn't reach the accuracy of the original model trained on the whole graph, the trade-off in accuracy is considered acceptable. For instance, each epoch of the partial retraining of the model on 15% of the user sessions (around 500 users) with approximately 40k interactions requires around 7 minutes of training. Consequently, the model converges quickly since it utilizes the pre-trained embeddings of the sessions in the training set, allowing for improving the network's performance without requiring arduous retraining.

Table 5.6: Summary of evaluation metrics of all methods.

Metric	Transductive	Offline Inference	Partial Retraining
Recall@5	0.8497	0.7942	0.8142
Recall@10	0.9098	0.8537	0.8761
3 NDCG@5	0.7529	0.67	0.6975
NDCG@10	0.7723	0.7072	0.732

This chapter outlined the results of the preprocessing steps and their impact on the dataset's structure, detailed the model's parameters and architecture, and evaluated its performance in various scenarios. The results demonstrate the model's high accuracy in predicting the next commands, particularly in dense interaction scenarios. Additionally, the offline inference method for new users presents a scalable solution for real-world applications, ensuring adaptability without prohibitive retraining costs.

Chapter 6

Conclusions and future works

This study introduced a pipeline that processes raw BIM event log files from Vectorworks to extract meaningful modeling sequences. The preprocessed data was then used to train a dynamic graph neural networks model for sequential recommendation. This method was enhanced to generate real-time predictions suitable for BIM production environments. The conclusions drawn from the results in the previous chapter and possible future improvements are described in this chapter.

6.1 Conclusions

Research questions stated in section 1.3 are answered in this section.

How can a robust pipeline be designed to preprocess event logs from BIM authoring tools and extract meaningful modeling sequences for next-command prediction?

To address the challenges posed by the unstructured nature of BIM event logs, we designed a comprehensive preprocessing pipeline tailored specifically to Vectorworks' event log files. This pipeline begins by cleaning the raw data, filtering out irrelevant events, and normalizing the data to ensure consistency. The pipeline then identifies and extracts sequences of logical modeling commands, which reflect the user's intent and workflow. We implemented techniques to group related actions into coherent sequences and remove redundancies to achieve this. This sequence extraction ensures that the input to predictive models is meaningful and interpretable. The pipeline was evaluated for the logical progression of extracted sequences and predictive performance metrics, such as accuracy, on the next-command prediction task. The results demonstrate that the pipeline effectively transforms noisy event logs into actionable data for sequential recommendation models.

How can GNNs be adapted to model sequential interactions in BIM environments for next-command prediction, and how do they compare with existing deep learning methods?

To explore the application of GNNs in BIM next-command prediction, we adopted a dynamic graph recommendation framework that captures the evolving preferences of users over time. GNNs were chosen for their ability to model complex dependencies and variable-length sequences, which are common in BIM workflows. Our implementation incorporates user and command interactions into a dynamic graph structure, enabling the model to learn user preferences effectively. The model predicts the next command based on historical user behavior by integrating sequence information and leveraging GNNs' representation learning capabilities. The results highlight GNNs' superior ability to handle variable-length sequences and adapt to dynamic changes in user behavior, making them a promising approach for BIM next-command prediction.

How effective is similarity-based inference in transferring pretrained user preferences to incoming users for a real-time next-command prediction in BIM?

One of the key limitations of traditional GNN-based recommendation systems is their inability to generalize to unseen graphs, particularly in the absence of initial node features. To overcome this challenge, we proposed a similarity-based inference mechanism that bridges the gap between training and retraining for new users. This approach aggregates information from pretrained user embeddings, weighted by similarity to the new user's initial interactions. By leveraging this aggregated representation, the model generates an embedding for the new user without requiring a complete retraining process. Experiments on simulated scenarios involving new users interacting with BIM tools showed that this method effectively predicts the next commands while maintaining computational efficiency. Although there is a trade-off in prediction accuracy, the approach significantly enhances the model's adaptability and scalability, making it suitable for real-world applications where new users frequently join the system.

What is the impact of scalability on accuracy in dynamic graph recommendation systems for BIM

In the context of dynamic graph recommendation systems, there is often a trade-off between achieving high prediction accuracy and ensuring scalability for practical applications.

Our similarity-based inference mechanism exemplifies this trade-off by allowing the model to generalize to new users without costly retraining. To evaluate the impact, we conducted experiments comparing the accuracy of this method to the traditional retraining approach. While a minor reduction in accuracy was observed, typically within 1-2%, the computational benefits were substantial, with the inference process being significantly faster and less resource-intensive. This trade-off was further analyzed regarding scalability, demonstrating that the proposed approach can efficiently handle a growing number of users and interactions. These findings suggest that the slight accuracy loss is a worthwhile compromise for the enhanced scalability and adaptability of the system, particularly in dynamic BIM environments.

6.2 Contributions

Our work on this thesis makes the following contributions:

- Since every BIM authoring tool is different in how it records its event log, we proposed a pipeline that processes Vectorwork's event log file, extracts the logical modeling sequence of the users, and trains and predicts the next modeling command.
- Although several research studies have used deep learning methods for BIM next command prediction, GNNs have never been tested in this area. We adopt a dynamic graph recommendation network to capture the user's evolving preference. We utilize GNN's advantage of dealing with variable sequences for user profile building and next command prediction based on user preferences.
- Most sequential recommendation GNN methods are transductive in the absence of initial node features. They cannot inherently generalize to new nodes. With our similarity-based inference, the gap between the initial training and retraining can be bridged in exchange for acceptable trade-offs in accuracy.

6.3 Limitations and future work

One of the limitations that was identified is that, despite the dataset being collected from a BIM authoring tool, the majority of recorded commands were similar to CAD commands, such as "Create Line", "Trim" and "Drag". The dataset does not contain many modeled objects of a particular discipline. The implemented method successfully achieved high accuracy in completing historical interaction sequences; however, it provides no insight into the relevance of the recommended command to the modeled object.

This is attributable to the absence of information about the modeled object. It may, therefore, be worthwhile to explore the implementation of an IFC logger alongside the event log files (Kouhestani, 2020) to record objects modeled and their respective locations, i.e., in which plan or view. This would create a parallel timeline that shows which objects were modeled in which time periods and where. Analyzing a group of commands within the same time period as an object provides insight into the user's task-specific preferences, facilitating more accurate short-term preference modeling.

Consequently, a graph network comprising three node types (user, command, object) can be constructed using this information. By considering the object attributes and creation time, the user profile (e.g., architect or structural/mechanical engineer), and the command description, it might be possible to construct an inductive framework capable of learning node representations for any graph. A tripartite graph can be constructed, as shown in figure 6.1, to offer more pathways to predict a user's next interaction/object (HSU & LI, 2021). The requisite dataset must be collected in a controlled environment where the modeled object is previously known, thereby ensuring the creation of log files specific to the task. This framework will combine collaborative filtering and content-based filtering to generate explainable recommendations.

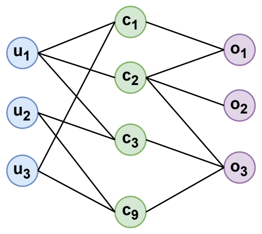


Figure 6.1: Tripartite graph construction, offering more pathways for message passing, hence more robust and task-specific recommendation (HSU & LI, 2021)

Another limitation that might be experienced when not enough data is available is circular predictions. When there is no sufficient data for a user, circular prediction (predicting popular commands often used by others) can lead to recommendations that lack novelty or relevance to the user's specific needs. This might also solved by incorporating user attributes and command features to create logical relations between users and commands. This might help recommend new commands that allow the user to find better commands relevant to the modeling sequence.

A further limitation is the hardware specifications employed to carry out the experiments. For instance, the combinations of GPU and RAM in the used setups didn't allow for experimenting on bigger datasets. The laptop had a better GPU (8 GB Nividia GeForce RTX 3070) but combined with only 16 GB RAM. The insufficient memory didn't allow for the full utilization of the GPU. On the other hand, the desktop used had a weaker GPU (5 GB) but combined with 32 GB RAM. Larger graphs could be loaded on the desktop, but longer training times were required due to weak GPU specifications. Moreover, the second-degree sub-graphs sampled before training required very large disk space. For example, a second-degree sub-graph sampling of 2500 sessions containing around 430,000 interactions and a maximum user and command length of 50 requires 24 GB of disk. As a result, huge disk spaces will be required to test higher sub-graph degrees or longer maximum sequence lengths.

Bibliography

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, *17*(6), 734–749. https://doi.org/10.1109/TKDE.2005.99

Andrew Shing-Tao Chang. (2002). Reasons for cost and schedule increase for engineering design projects. *Journal of Management in Engineering*, *18*(1), 29–36. https://doi.org/10.1061/(ASCE)0742-597X(2002)18:1(29)

Boka, T., Niu, Z., & Neupane, R. (2024). A survey of sequential recommendation systems: Techniques, evaluation, and future directions. *Information Systems*, *125*, 102427. https://doi.org/10.1016/j.is.2024.102427

Borrmann, A., König, M., Koch, C., & Beetz, J. (2018). *Building information modeling : Technology foundations and industry practice*. Springer International Publishing AG. http://ebookcentral.proquest.com/lib/munchentech/detail.action?docID=5520323

Brebbia, C. A., & Collins, M. W. (2004). *Design and nature ii: Comparing design in nature with science and engineering* (Vol. 6). WIT.

Bundesministerium für Verkehr und digitale Infrastruktur. (2015). Stufenplan digitales planen und bauen: Einführung moderner, it-gestützter prozesse und technologien bei planung, bau und betrieb von bauwerken.

Company, M. (2020). Adapting to the next normal in retail: The customer experience imperative.

Corso, G., Stark, H., Jegelka, S., Jaakkola, T., & Barzilay, R. (2024). Graph neural networks. *Nature Reviews Methods Primers*, *4*(1), 17.

Du, K.-L., & Swamy, M. N. S. (2019). *Neural networks and statistical learning* (Second edition). Springer. https://zbmath.org/?q=an%3A07098306

eMarketer. (2014). Global b2c ecommerce sales to hit $1.5 trillion this year driven by growth in emerging markets.

FULFORD, R., & STANDING, C. (2014). Construction industry productivity and the potential for collaborative practice. *International Journal of Project Management*, *32*(2), 315–326. https://doi.org/10.1016/j.ijproman.2013.05.007

GAO, W., WU, C., HUANG, W., LIN, B., & SU, X. (2021). A data structure for studying 3d modeling design behavior based on event logs. *Automation in Construction*, *132*, 103967. https://doi.org/10.1016/j.autcon.2021.103967

GAO, W., ZHANG, X., HE, Q., LIN, B., & HUANG, W. (2022). Command prediction based on early 3d modeling design logs by deep neural networks. *Automation in Construction*, *133*, 104026. https://doi.org/10.1016/j.autcon.2021.104026

GOODFELLOW, I. (2016). Deep learning.

HE, X., LIAO, L., ZHANG, H., NIE, L., HU, X., & CHUA, T.-S. (2017). Neural collaborative filtering. *Proceedings of the 26th international conference on world wide web*, 173–182.

HERAKOVIC, N., ZUPAN, H., PIPAN, M., PROTNER, J., & SIMIC, M. (2019). Distributed manufacturing systems with digital agent. *65*, 650–657. https://doi.org/10.5545/sv-jme.2019.6331

HERLOCKER, J., KONSTAN, J., BORCHERS, A., & RIEDL, J. (1999). An algorithmic framework for performing collaborative filtering.

HIDASI, B., KARATZOGLOU, A., BALTRUNAS, L., & TIKK, D. (2016). Session-based recommendations with recurrent neural networks. https://arxiv.org/abs/1511.06939

HSU, C., & LI, C.-T. (2021). Retagnn: Relational temporal attentive graph neural networks for holistic sequential recommendation, 2968–2979. https://doi.org/10.1145/3442381.3449957

HUANG, X., QIAN, S., FANG, Q., SANG, J., & XU, C. (2018). Csan: Contextual self-attention network for user sequential recommendation. *Proceedings of the 26th ACM International Conference on Multimedia*, 447–455. https://doi.org/10.1145/3240508.3240609

JANG, S., LEE, G., SHIN, S., & ROH, H. (2023). Lexicon-based content analysis of bim logs for diverse bim log mining use cases. *Adv. Eng. Inform.*, *57*(100). https://doi.org/10.1016/j.aei.2023.102079

JONES, S., & LAQUIDARA-CARR, D. (2021). Accelerating digital transformation through bim: Regional focus: Germany, 1.

KANG, W., & MCAULEY, J. J. (2018). Self-attentive sequential recommendation. *CoRR*, *abs/1808.09781*. http://arxiv.org/abs/1808.09781

KOREN, Y. (2008). Factorization meets the neighborhood: A multifaceted collaborative filtering model. *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 426–434. https://doi.org/10.1145/1401890.1401944

KOREN, Y. (2010). Collaborative filtering with temporal dynamics. *Commun. ACM*, *53*(4), 89–97. https://doi.org/10.1145/1721654.1721677

KOREN, Y., BELL, R., & VOLINSKY, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, *42*(8), 30–37. https://doi.org/10.1109/MC.2009.263

KOREN, Y., RENDLE, S., & BELL, R. (2021). Advances in collaborative filtering. https://doi.org/10.1007/978-1-0716-2197-4{\textunderscore}3

LI, J., WANG, Y., & MCAULEY, J. (2020). Time interval aware self-attention for sequential recommendation, 322–330. https://doi.org/10.1145/3336191.3371786

LI, J., REN, P., CHEN, Z., REN, Z., & MA, J. (2017). Neural attentive session-based recommendation. https://arxiv.org/abs/1711.04725

LIN, J.-R., HU, Z.-Z., ZHANG, J.-P., & YU, F.-Q. (2016). A natural-language-based approach to intelligent data retrieval and representation for cloud bim. *Computer-Aided Civil and Infrastructure Engineering*, *31*(1), 18–33. https://doi.org/10.1111/mice.12151

LIU, Q., ZENG, Y., MOKHOSI, R., & ZHANG, H. (2018). Stamp: Short-term attention/memory priority model for session-based recommendation. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1831–1839. https://doi.org/10.1145/3219819.3219950

LIU, Y., XIA, L., & HUANG, C. (2024). Selfgnn: Self-supervised graph neural networks for sequential recommendation. *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1609–1618. https://doi.org/10.1145/3626772.3657716

LUO, T., LIU, Y., & PAN, S. J. (2024). Colluo2024collaborativelaborative sequential recommendations via multi-view gnn-transformers. *ACM Transactions on Information Systems*.

MA, C., MA, L., ZHANG, Y., SUN, J., LIU, X., & COATES, M. J. (2019). Memory augmented graph neural networks for sequential recommendation. https://api.semanticscholar.org/CorpusID:209501162

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115–133. https://doi.org/10.1007/BF02478259

Mienye, I. D., Swart, T. G., & Obaido, G. (2024). Recurrent neural networks: A comprehensive review of architectures, variants, and applications. *Information*, *15*(9). https://doi.org/10.3390/info15090517

Oyewobi, L., & Ogunsemi, D. (2010). Factors influencing reworks occurrence in construction: A study of selected building projects in nigeria. https://api.semanticscholar.org/CorpusID:107452913

Pan, Y. (2021). *Mining building information modeling (bim) event logs for improved project management*. https://doi.org/10.32657/10356/152484

Pan, Y., & Zhang, L. (2020a). Bim log mining: Learning and predicting design commands. *Automation in Construction*, *112*, 103107. https://doi.org/10.1016/j.autcon.2020.103107

Pan, Y., & Zhang, L. (2020b). Sequential design command prediction using bim event logs, 306–315. https://doi.org/10.1061/9780784482865.033

Paterek, A. (2007). Improving regularized singular value decomposition for collaborative filtering. *Proceedings of KDD Cup and Workshop*.

Peng, Y., Lin, J., Zhang, J., & Hu, Z. (2017). A hybrid data mining approach on bim-based building operation and maintenance. *Building and Environment*, *126*, 483–495. https://doi.org/10.1016/j.buildenv.2017.09.030

Qiu, R., Huang, Z., Yin, H., & Wang, Z. (2022). Contrastive learning for representation degeneration problem in sequential recommendation. *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, 813–823. https://doi.org/10.1145/3488560.3498433

Radnia, A. (2021). Sequence prediction applied to bim log data, an approach to develop a command recommender system for bim software application.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, 175–186. https://doi.org/10.1145/192844.192905

Ricci, F., Rokach, L., & Shapira, B. (2010). Introduction to recommender systems handbook. In *Recommender systems handbook* (pp. 1–35). Springer.

Russel, P., & Elger, D. (2008). The meaning of bim: Towards a bionic building. *Architecture in Computro [26th eCAADe Conference Proceedings]*, 531–536.

Salakhutdinov, R., & Mnih, A. (2007). Probabilistic matrix factorization, 1257–1264.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th International Conference on World Wide Web*, 285–295. https://doi.org/10.1145/371920.372071

Schafer, J. B., Konstan, J., & Riedl, J. (1999). Recommender systems in e-commerce. *Proceedings of the 1st ACM Conference on Electronic Commerce*, 158–166. https://doi.org/10.1145/336992.337035

Schmidt, R. M. (2019). Recurrent neural networks (rnns): A gentle introduction and overview. *arXiv preprint arXiv:1912.05911*.

Srivastava, J., Cooley, R., Deshpande, M., & Tan, P.-N. (2000). Web usage mining: Discovery and applications of usage patterns from web data. *SIGKDD Explor. Newsl.*, *1*(2), 12–23. https://doi.org/10.1145/846183.846188

Staudemeyer, R. C., & Morris, E. R. (2019). Understanding lstm–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*.

Sun, F., Liu, J., Wu, J., Pei, C., Lin, X., Ou, W., & Jiang, P. (2019). Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. *CoRR*, *abs/1904.06690*. http://arxiv.org/abs/1904.06690

van der Aalst, W. M. P. (2016). *Process mining: Data science in action* (2nd ed. 2016). Springer Berlin Heidelberg; Imprint: Springer.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. https://arxiv.org/abs/1710.10903

Vrahatis, A. G., Lazaros, K., & Kotsiantis, S. (2024). Graph attention networks: A comprehensive review of methods and applications. *Future Internet*, *16*(9), 318.

Wang, X., He, X., Wang, M., Feng, F., & Chua, T.-S. (2019a). Neural graph collaborative filtering, 165–174. https://doi.org/10.1145/3331184.3331267

Wang, X., He, X., Wang, M., Feng, F., & Chua, T.-S. (2019b). Neural graph collaborative filtering. *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, 165–174.

Wang, Z., Wei, W., Cong, G., Li, X.-L., Mao, X.-L., & Qiu, M. (2020). Global context enhanced graph neural networks for session-based recommendation, 169–178. https://doi.org/10.1145/3397271.3401142

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., & Weinberger, K. (2019). Simplifying graph convolutional networks. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning* (pp. 6861–6871). PMLR. https://proceedings.mlr.press/v97/wu19e.html

Wu, J., Hu, C., Wang, Y., Hu, X., & Zhu, J. (2020). A hierarchical recurrent neural network for symbolic melody generation. *IEEE Transactions on Cybernetics*, *50*(6), 2749–2757. https://doi.org/10.1109/TCYB.2019.2953194

Wu, L., Li, S., Hsieh, C.-J., & Sharpnack, J. (2020). Sse-pt: Sequential recommendation via personalized transformer, 328–337. https://doi.org/10.1145/3383313.3412258

Wu, S., Tang, Y., Zhu, Y., Wang, L., Xie, X., & Tan, T. (2019). Session-based recommendation with graph neural networks. https://doi.org/10.1609/aaai.v33i01.3301346

Zhang, M., Wu, S., Yu, X., Liu, Q., & Wang, L. (2022). Dynamic graph neural networks for sequential recommendation. *IEEE Transactions on Knowledge and Data Engineering*, *35*(5), 4741–4753.

Zheng, Y., Liu, S., Li, Z., & Wu, S. (2020). Dgtn: Dual-channel graph transition network for session-based recommendation, 236–242. https://doi.org/10.1109/ICDMW51313.2020.00041

Zhu, Y., Li, H., Liao, Y., Wang, B., Guan, Z., Liu, H., & Cai, D. (2017). What to do next: Modeling user behaviors by time-lstm, 3602–3608.

Declaration

I hereby affirm that I have independently written the thesis submitted by me and have not used any sources or aids other than those indicated.

Location, Date, Signature