SEQUENCE PREDICTION APPLIED TO BIM LOG DATA, AN APPROACH TO DEVELOP A COMMAND RECOMMENDER SYSTEM FOR BIM SOFTWARE APPLICATION

by

Ashkan Radnia

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
degree of Master of Architecture and
Master of Science in Information Technology

Charlotte

2021

Approved by:

_____

Prof. Jefferson Ellinger

_____

Dr. Minwoo Jake Lee

_____

Dr. Samira Shaikh

ABSTRACT

ASHKAN RADNIA. SEQUENCE PREDICTION APPLIED TO BIM LOG DATA,
AN APPROACH TO DEVELOP A COMMAND RECOMMENDER SYSTEM
FOR BIM SOFTWARE APPLICATION
(Under the direction of PROF. JEFFERSON ELLINGER)

Building Information Modeling has become an industry standard for designing, documenting, and collaborating within the architecture, engineering, and construction industry. Architects spend most of their time developing projects from the feasibility study phase through post occupancy operation using BIM software such as Autodesk Revit. There are many tools created to automate tedious tasks or speed up the design process, however, most tasks require manual work, therefore taking longer to complete and increasing the probability of design errors. With each new release software developers are adding more features and tools to increase application capabilities; however, these are often underutilized since the proper training and resources for adopting the tools are not always available. Due to this limitation, designers will tend to stick to the tools they are already comfortable that are not necessarily best fit for all tasks.

This thesis investigates using Autodesk Revit Journals as a non-intrusive method to extract sequential data about user interactions with the software and use collected data to conduct analysis on design and work patterns of users. Additionally, collected data is used to develop a neural network that takes sequence of user commands and trains to predict the next command. The developed neural network can serve as a recommender system suggesting the most suitable commands for the user, enabling improved and efficient workflows and enforcement of best modeling practices.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

BIM	Building Information Modeling
AEC	Architecture Engineering Construction
ML	Machine Learning
ANN	Artificial Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
GRU	Gated Recurrent Unit
HCI	Human Computer Interaction
UI	User Interface

INTRODUCTION

BACKGROUND

The architectural design process has changed drastically in the past decades. As technology has advanced, numerous tools and software programs have become an integral part of an architect's day-to-day work. These programs increase work efficiency, reduce errors in the design process, and have made possible the design and construction of some of the most influential architectural designs that were not imaginable prior to the emergence of computers in offices.

In the past decades, beginning with the early 70's, there has been a gradual transition from manual drafting tools such as T-squares and compasses to digital software applications such as Computer Aided Drawing. As a digital substitute for manual drafting tools, CAD allowed for faster development of designs and only required one operator to produce drawings. This transition drastically reduced the cost of errors and sped up the design process. However, there were still many limitations because the designer still had to produce single drawings and documents for each part of the project. If any changes were to be made, all drawings and documents had to be manually updated to reflect that change.

Decades later, Building Information Modeling emerged to disrupt 2D CAD-based workflows. The architecture, Engineering, and Construction industry have widely adopted this technology over the past two decades. BIM is believed to be one of the most promising advancements in the industry because it can transform and enhance performance by decreasing inefficiencies, improving productivity, and increasing collaboration among project stakeholders.

[1]. In addition to the capabilities of 3D CAD, the virtual building model connects to a database, storing all necessary information about individual components that go into the building's construction and operation. This data-rich framework of building design helps architects, engineers, and contractors design, document, and collaborate simultaneously. [2]] It offers visualization of design, fast creation of alternatives, automatic examination of model reliability, production of reports, and building performance forecasting. [3]. Furthermore, BIM applications are not limited to the construction phase; they extend to post-occupancy operations and provide helpful information at all building stages.

In 2015, 79% of construction projects adopted BIM in the United States [4]. Since the release of Autodesk Revit in 2000, it has seen exponential growth in users, and by 2019, it had a 46% share of the global BIM software market. Employers highly demand Revit skills since proficiency in using it will reduce the deficiency and decrease design errors which are among the

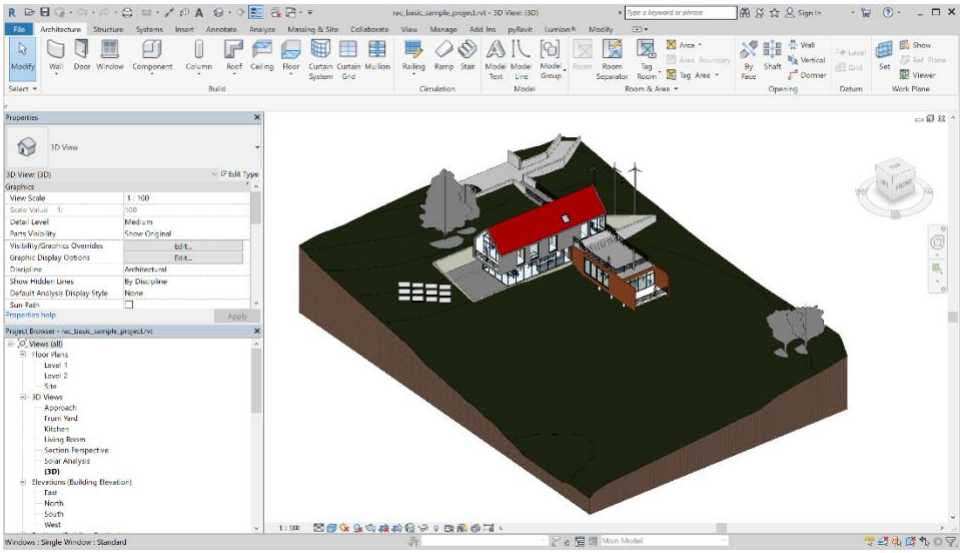

Figure 1: Autodesk Revit User Interface

major risk factors causing rework and delay [5].

The growing influence of computational design has made automation and generative tools ubiquitous among every firm's software suits. An Architect spends a substantial portion of their time designing, editing, and fixing errors using Revit. Designers utilize such tools to improve their workflows by automating tedious tasks, developing designs with complex geometry, or finding solutions that require the consideration of multiple criteria. Design is a complex and parallel process; hence, it requires a lot of exploration before arriving at a final design decision. Despite the substantial efforts to improve and promote the use of computational tools, designers often have a hard time integrating them into their workflow due to the shortcomings of these tools to accommodate the complex design process. With manual labor being a reality in the design process, extensive training, troubleshooting, user education, and continuous control of design and software errors are needed. By integrating tools that assist with the manual process into the overall workflow helps to reduce errors and increase efficiency without reducing flexibility.

BIM software applications offer many commands for performing various tasks, and with each release, additional features are added. It is arduous for managers to keep users up to date, enforce best modeling practices, and maintain a healthy model to prevent errors. If not appropriately managed, software errors can significantly reduce the modeling process's speed and increase the model size [6]. The challenges mentioned above contribute to making AEC one of the industries with the least efficiency and productivity, lagging other industries in terms of progress in efficiency [7], [8].

This research seeks to analyze the manual design process's characteristics by non-intrusively collecting and analyzing data from user interactions across multiple projects derived from Revit logs called Journals. A Journal collection provided by an international architecture

firm was analyzed and processed to identify and extract relevant and valuable information about sequences of user interactions with the software during their design process. The processed data was then used to develop a Machine Learning algorithm to suggest commands to the user as they work within the software and execute commands. Ultimately, this system will help improve the user experience within the complex interface of Revit and increase their modeling efficiency.

Machine learning (ML) has been growing exponentially in the past years due to unprecedented access to data, increased hardware capabilities, and unforeseen developments in ML tools that make these advanced and complex techniques more accessible for a wide range of users. ML techniques are being applied extensively to solve problems and automate mundane tasks across a variety of domains. Contrary to the exponential growth and adoption of ML in other industries, the AEC industry has lagged in the uptake of ML solutions for addressing the challenges of the industry [9]. BIM platforms can be leveraged to extract valuable data and develop ML applications using them to address many of the industry's challenges.

We explored the use of machine learning approaches to find methods and workflows to leverage Revit Journals to improve the working experience for architects. A supervised ML model for a sequence prediction was developed and trained on the collected data from Journals to predict the next command sequence. This model can serve as a recommender system, helping users to have improved workflows and enforce best modeling practices by providing useful suggestions. Unsupervised models, such as topic modeling, were also explored to learn more about similar commands and usage patterns across multiple users and projects.

LITRETURE REVIEW

In computing, logs are referred to as files that record events that occur in an operating system, other software runs, or any information about processes during the execution of a system or software. Logs have several characteristics that make them ideal for research on user interface design and the interaction between humans and computers. Each time an application is used, Logs can be used to collect data on any number of users over time. [10].

Log files can be analyzed to understand user behaviors across different platforms. With extensive usage of web applications, a large amount of log data gets collected and stored. The collected data is then used to perform a wide range of analyses and studies, such as user behavior prediction, product recommendation, optimized hardware distribution, and platform enhancements for optimized user experience. [11], [12]. One example of using logs is in E-commerce platforms, which rely heavily on log data. As these platforms grow in popularity, it is vital for them to predict users' purchase and browsing behavior to maximize their sales, manage inventory, and increase user engagement. In “A Method of Purchase Prediction Based on User Behavior Log” by Li et al., a method using probability statistics was used to predict user's purchase preferences based on their click patterns gathered from a well-known international mobile e-commerce platform [13]. Log files are also used for generating recommendations based on their purchasing behavior and item browsing history and recommend item bundles that are more likely to be bought together [14]

Another example where log data is being used is in Human-computer Interaction (HCI) research. In HCI, researchers utilize log files to study the behavioral traits of users. Log files are rich data sources because they can collect a large scale of data from people interacting with systems in the most natural way. [15] Using log files, HCI researchers can categorize user website

revisiting patterns [16], creating an interface that facilitates information re-use from previously visited emails, web pages, documents, and appointments.

Working with log files is challenging since they store a large volume of unstructured data that requires a lot of effort to identify valuable information from the raw logs and then extract and structure them. Some of the major errors and distortions in the log are missing events, dropped data, and misplaced semantics. These issues must be addressed to get realistic and valuable information through exploratory data analysis and pre-processing before using the data for further research. [15]

A few studies have been conducted on BIM logs. In one study, operation data from Revit logs files, also known as a Journal, was analyzed using "Rapid Miner" software to understand the 3D building modeling process. The data used was information about general and specific commands that the user had executed in developing a framing model for a typical two-story building. The study's goal was to identify the most used commands and give suggestions to improve them [17]. A similar approach was taken to mine implicit 3D modeling patterns from a dataset. Log files of multiple Revit projects from an international firm were collected and parsed to extract data about users, project names, commands, etc. A sequence mining algorithm based on Generalized Suffix Trees (GST) was used on the extracted data to identify implicit 3D modeling patterns and the total execution time of that pattern for different users. A significant average time difference for executing identical commands was observed among various users showing differences in working styles [18]. In another similar research, Zhang et al. used PATRICIA (PAT) tree algorithm to mine and characterize patterns in the design log dataset. They studied the frequency of patterns and the time that it takes each user to execute one of the sequences as a metric for productivity measurement. It was discovered that one particular pattern

(Select to modify - Trim Extend – Finish Sketch) was accounting for 46% of instances associated with the top five discovered sequential patterns of design commands. [8].

Pan et al. used a Long Short Term Memory neural network on structured data extracted from log files to predict the class of the following possible command in a sequence as the next step in the design process's automation. They manually categorized commands in 14 distinct categories and trained their neural network on the commands to classify sequence. The trained model achieved 70% accuracy outperforming other methods such as the K-nearest neighbor machine learning algorithm. [19].

Most of the studies that used BIM log files for knowledge discovery and analysis had similar approaches and workflows in working with the data. The first step is extracting useful information from the raw log files and then structuring them as tabular data.

The task of recommender systems is to turn data on users and their preferences into predictions of probable future likes, interests, and actions. [20] Today recommender systems and computer-generated suggestions are an inseparable part of our daily digital experience. These systems help us finish a sentence when writing an email, select what content to choose to entertain ourselves, or suggest what we should buy for our next purchase. Therefore, it is crucial for businesses to develop a sophisticated algorithm to match user preferences.

Machine Learning and Deep Learning have become widely adopted to solve some of the most challenging problems across different disciplines. ML models use algorithms and computational statistic to learn from data without being explicitly programmed. ML application's exponential boom has been enabled by easy access to massive volumes of data and increased hardware capabilities. Deep learning-based recommender systems are one of the most successful

applications of data mining and machine-learning technology in practice that are proliferating. They provide higher quality results compared to traditional methods such as collaborative filtering. Sequence-aware recommendation systems developed with deep learning algorithms such as Recurrent Neural Networks, which uses data from sequential logs, can use both long-term and short-term histories of user sequential actions to recommend items that match those patterns. Some application examples of RNN based recommender systems are click prediction for online advertisement, prediction of the next item in a sequence given the current one [21], and modeling user activity in e-commerce platforms based on sessions [22].

Recommendation systems applications are not limited only to content and item recommendation within web-based applications and mobile apps. Soh et al. implemented a deep sequential recommendation system that suggests a personalized user interface based on the latent embedding of parameters. The Adaptive User Interface can enhance the usability of complex software by providing real-time contextual adaptation and assistance.[23].

Command recommendation systems in the context of complex software applications can aid users in enhanced utilization of programs and suggest tools that are a good fit while they are working. These suggestions also bring awareness to the tools that are available that often go unnoticed. Two main approaches exist for developing such systems, Social and Task-Based. The social approach uses data collected from the software usage logs and applies deep learning or collaborative filtering methods to make recommendations. On the other hand, the task-based approach mines user manuals and documentation and applies logical clustering for command groups. [24]. Autodesk AutoCAD, one of the most used CAD drawing software, has hundreds of different commands, and it continues to grow more with each updated version. However, the largest group of its users only use between 31-40 commands. A research group from Autodesk

developed a recommender system, Community Commands, using a dataset obtained from participants in the Customer Involvement Program with 40 million observations from 16,000 users culminated over a 6-month period. The recommender system was developed using collaborative filtering and was made accessible to the users via a window within AutoCAD. After the user installs the system, it collects user commands history for some time. Once the collection period ends, it suggests the top 10 most relevant commands to the user.[25]

IMPLEMENTAION

WORK OUTLINE

Two steps were involved in implementing this research; the first step was to analyze, discover, and comprehend how user events and data are recorded in a journal, and the second step was to develop a pipeline to extract the information and structure it as a tabular data frame.

Raw log files were analyzed to extract string patterns and develop a script to parse raw Journals to collect data and store them as a structured data frame. Different exploratory data analysis was performed on the collected dataset to gain insight into data for feature extraction and preparation for Machine Learning development. This analysis provided necessary insights to decide what features and observations should be kept and identify null, irrelevant, or noise data to be removed. This process also helps us learn about key features, the distribution of event instances, and underlying implicit structures and patterns.

The second step was to develop the Machine Learning model, which began by formulating the task and then selecting a model architecture that was a good fit for the task and the available data. The task for this research was defined as a supervised machine learning problem, which is fed with inputs and learns to map inputs to labels. Inputs were created as a sequence of commands, and the labels were the following actions in the sequences. Because of our dataset's sequential and temporal type, a neural network architecture capable of capturing this type of relation was selected. As we trained and evaluated the model throughout the development process, we tuned model parameters and tried different data configurations to improve the model's predictions.

In the subsequent chapters, the work process for each step will be explained further, and implementation details, challenges, failures, and achievements will be discussed.

DATA COLLECTION AND PREPRATION

Journal files are unstructured timestamped log files that Autodesk Revit generates for the working sessions. Capturing information begins once Revit starts, and it is saved to the Journal directory on the system. The Journal records information about the system, file paths, external loaded add-ins and plugins, user information and modeling activities, event timestamps, errors, warnings, failures, etc. During each phase of a project, multiple log files with numerous lines of information are generated and are often discarded after a brief period since they are mainly used for file recovery, diagnostics, and troubleshooting by Autodesk customer support once a project file gets corrupt.

The resources available about Journal files are limited, and they do not give enough information about how events are recorded in it and which keywords are used when recording different event types. To extract information from the Journal and structure it, we need to identify the main keywords and patterns Revit uses to record this information in Journals.

We used the extracted string patterns to make Regular Expression search strings to parse the log files. The strategy for understanding Journals was to start a blank Revit project and execute a few commands with a defined order. Then, we closed the Revit, browsed the Journal directory, and performed a simple string search on the session's Journal for keywords we know, such as project names. This method allows us to identify and extract string patterns from the large Journal file and record any keywords or string patterns that we find essential. A large international architecture firm provided the Journal archive for this research which was collected over six months period.

We already knew the Username for the sample Journal is "aradnia," and the file name is "House Model.rvt", we searched for matching strings in the sample Journal and identified Revit records username as "Jrn. Directive "Username" _" and file name uses a similar pattern as "Jrn.Directive "DocSymbol." " Events that users produce are saved with "Jrn." prefix. After verifying the patterns with some samples from the Journal archive, we continued to apply the same method for extracting other information such as commands, Revit version, etc. Error events are recorded similarly with the keyword "Jrn.Data "Error dialog" _" followed by detailed information about error type (failure, error, warning) and the number of errors raised.

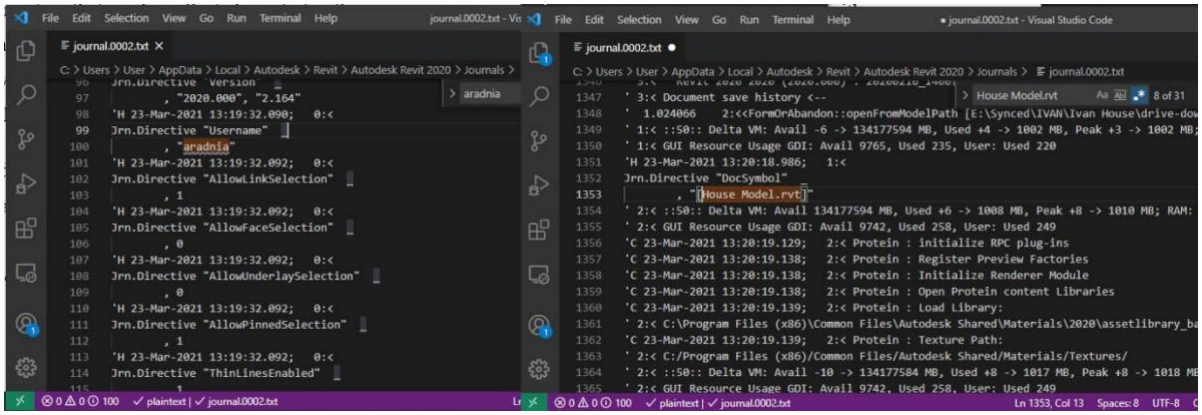

Figure 2: Revit Journal and String patterns for Username and File name

After manually identifying string patterns for different events, we used Regular Expression to parse the Journals archive. Regular Expression, often referred to as RegEx is a sequence of characters that specifies a search pattern. String searching algorithms use these patterns for different operations on the strings. We ran the parser script on the Journal archive and extracted data. usernames and project name were assigned to observation based on the last recorded Journal entry before that Event. For instance, if we collect "Jrn.Directive "Username" _ A" username "A" was assigned to all the commands that came after that observation. If a change occurs in a Journal such as "Username" _ B" user B is registered for the commands from thereon.

Once the parsing process was done, all the extracted information was stored as a data frame and exported with “csv” file extensions to be used in the next steps.

EXPLORATORY DATA ANALYSIS

The most important part of developing a successful machine learning model is to have adequate, high-quality, and clean data. Data parsed from Journals was processed and analyzed so it can be used for training the model. Processing is done to handle null values and noise in the observation and standardization. The dataset consists of 26,132,122 observations with six categorical features and two date-time features. To get familiarized with the data and the features, we performed exploratory data analysis. We produced different data visualizations to help uncover implicit and explicit patterns and identify key features and values. Insights from this step were used to refine our dataset and extract features for developing the machine learning model.

Event Properties	count	unique	top	freq
UserName	25711077	221	caterinac	956412
FileName	25711077	28247	19123CS-ARCH-B2-WIP_yiep.rvt	295652
EventName	25711077	7	CommandTrack	25506040
Access	25560261	60	AccelKey	17802360
Event	25560261	1584	ID_CANCEL_EDITOR	15468859
RevitYear	25711077	6	2020	16872639
EventEndTime	25711077	25410351	2020-10-20 14:30:27.093	120
FileType	25711077	3	RevitProject	25211817

Figure 3: An overview of collected data

The analysis began by looking at the relationship between the time and the observations. Figure 5 illustrates the total number of events executed on each day of the week, and Figure 4 shows the number of commands executed and warnings or errors raised for each hour of the day. There are fewer events recorded over the weekend, and a positive correlation exists between the

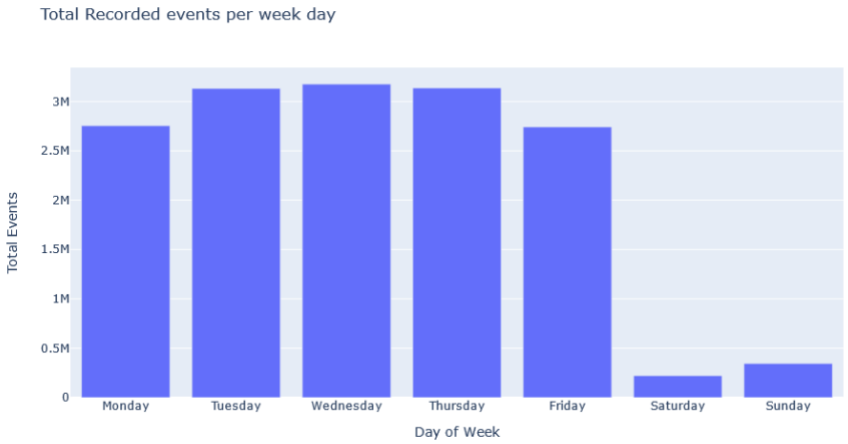

Figure 5 : Total Recorded events for each day of week.

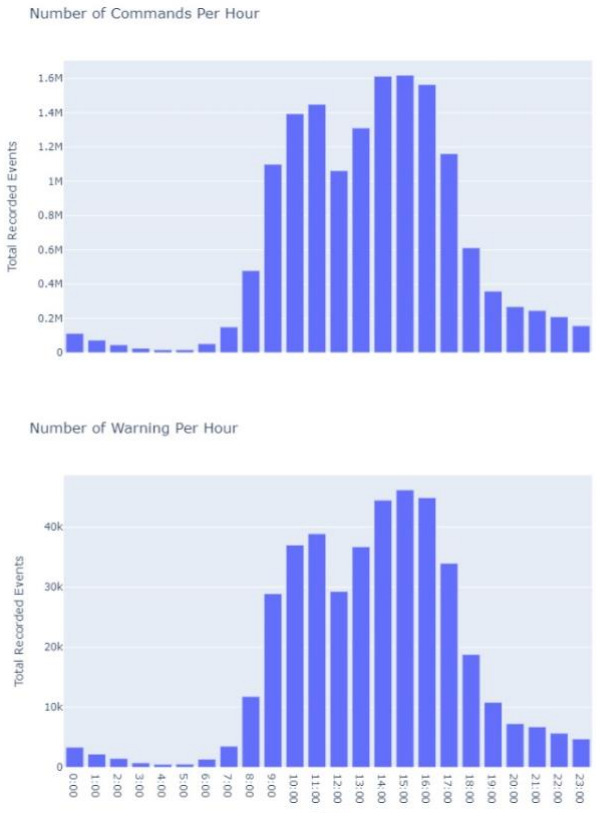

Figure 4: Number of commands and warnings per hour

number of executed commands and errors. In a workday, the number of commands performed increases steadily for at least the first half of the day, then drops at noon, and then we see a steady increase again in the afternoon.

Each user tends to utilize specific tools, working on different tasks and developing unique work routines. By looking at the top 10 executed commands for the six users with the highest recorded events, we can see each of them has different command usage patterns, but some commands such as ID_ALIGN seem to be mutual between all of them (Figure 6).

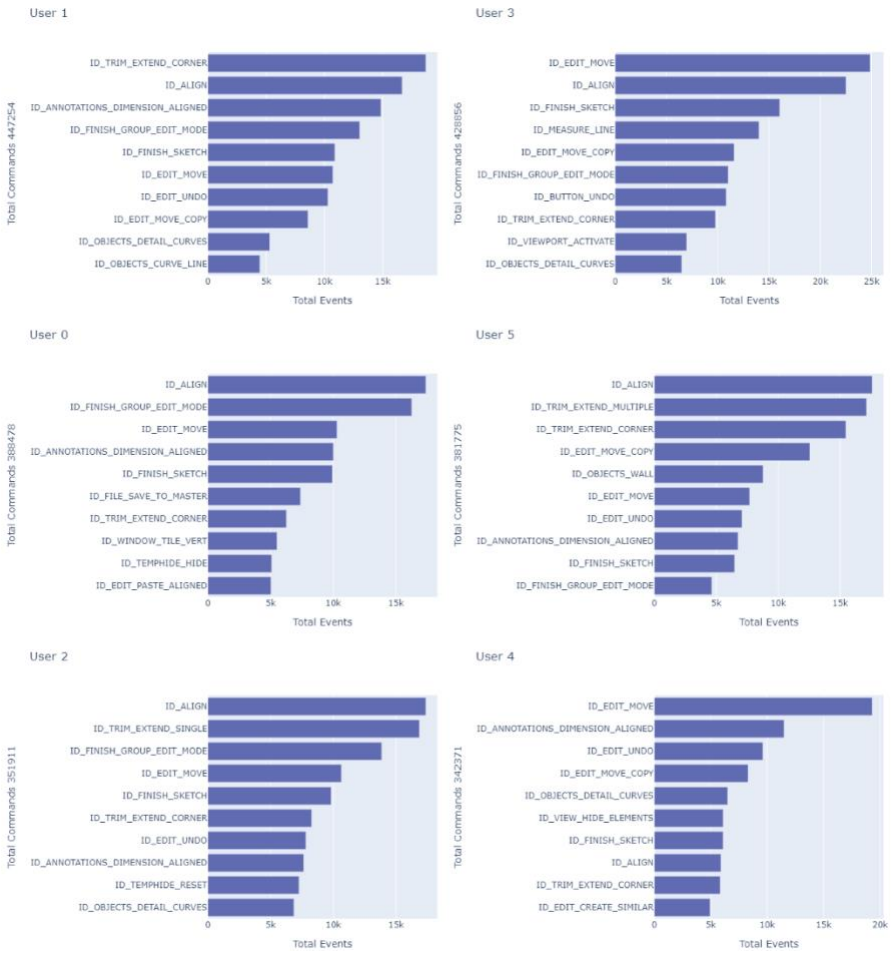

Figure 6: Top 10 most used command of users with highest number events executed

Each project also has different specifications; therefore, different requiring diverse types of tools and workflows for developing their design. Figure 7 illustrates the most used commands for the projects that had the highest number of events recorded. Like top users' command frequency, the

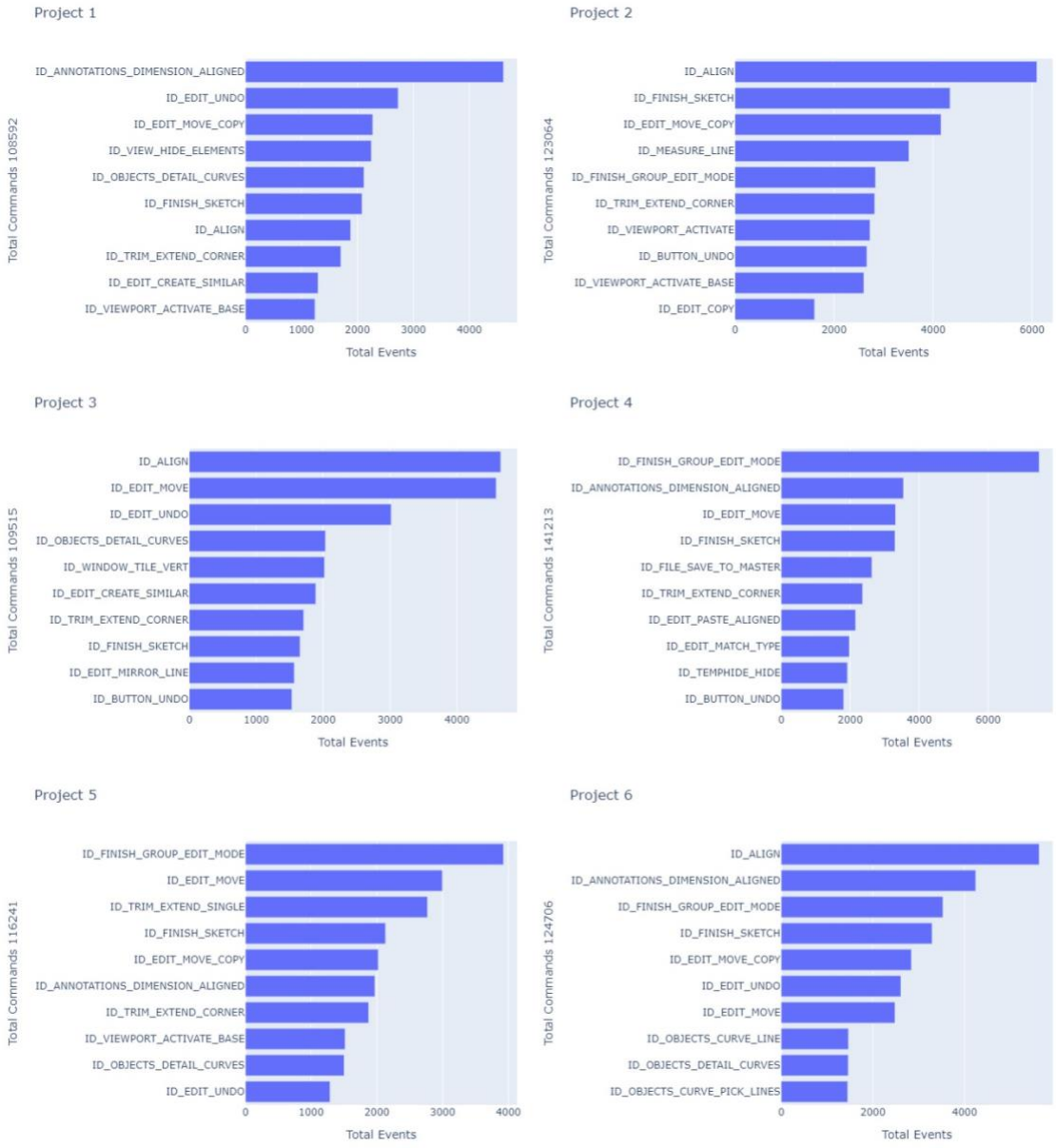

Figure 7: Top commands for top 6 projects with the most recorded events.

most frequently used commands for each project are different.

To better understand the order of commands, we created a graph network based on event sequences using network-x an open-source python library. A node was defined for every unique event, and bi-directional edges between nodes were added whenever an event followed another one. Edge weights were calculated by the number of times that the pattern was observed. The

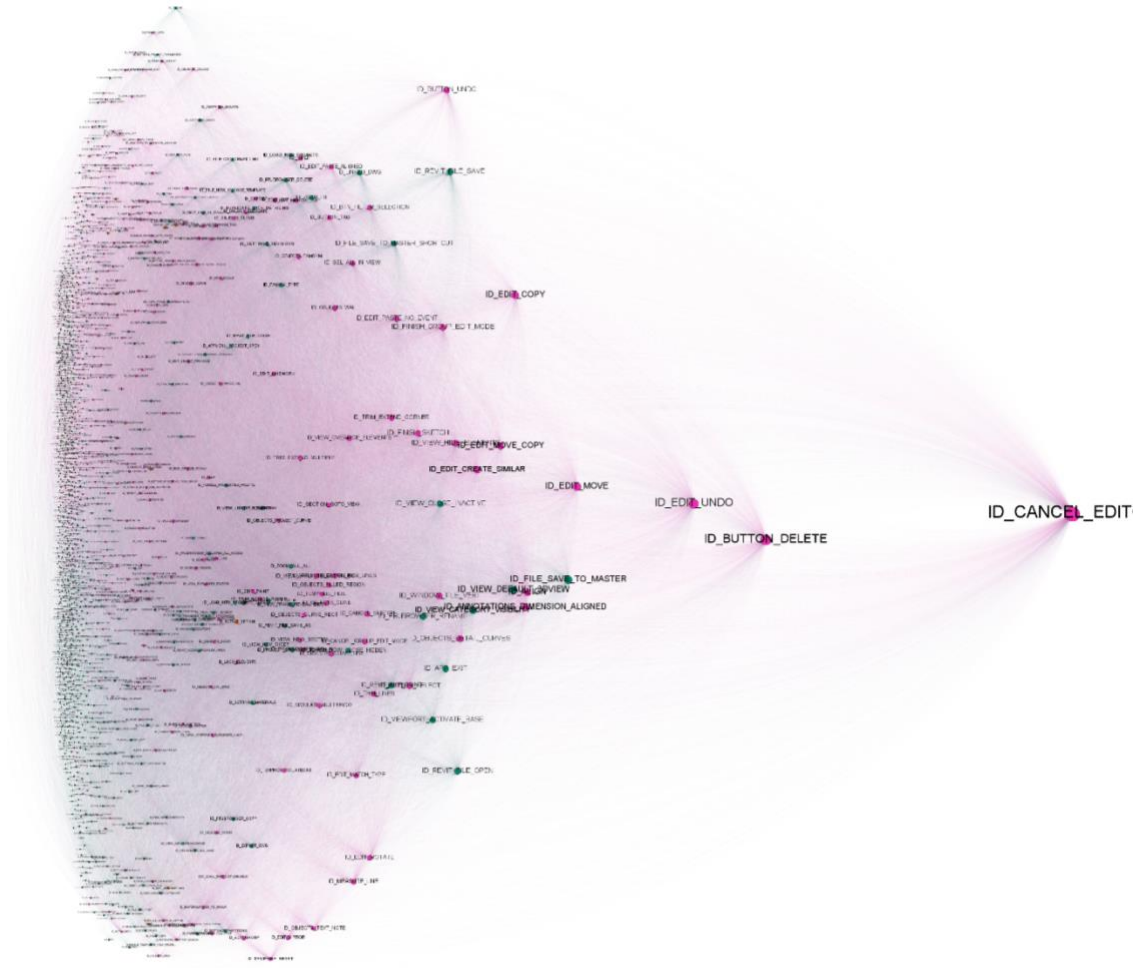

Figure 8: Command's network constructed from ordered sequences in the dataset. Nodes are Command ID and edges are defined if they were executed before or after each other. The color of nodes and edges correspond to Community detection results and node

created network was taken to Gephi [26] to visualize and analyze the network based on communities and edge weights, and other calculated network properties. Modularity class was calculated based on the nodes and their weights, and it groups nodes used together more frequently. We used to calculate the modularity class to assign colors to clusters. Node sizes and

the order from right to left indicate how often that command was executed after other commands, with most frequent having the largest node size and being placed on the right side of the diagram. Figure

The insights gained from the exploratory data analysis process will help us to curate and refine data for each experiment to improve results. We did a general refinement to do a preliminary data cleaning and normalization and removed features or values that are missing, redundant, or cannot be used for predicting the target features. Observations with missing features or irrelevant data were removed. To normalize the distribution of observation, those with extremely high or low frequencies were removed to prevent imbalanced data. The event feature has 1,584 unique values with frequencies ranging from 1 to 15,468,859, and "ID_CANCEL_EDITOR" had the highest frequency, with 6 million observations more than the second most frequent Event "ID_CANCEL_DELETE." The high frequency of these events can be explained by the way Revit users work. Whenever users want to select a new tool or change process within Revit, they press the Escape button, and pressing it repeatedly becomes a habit. Both ID_CANCEL_EDITOR and ID_BUTTON_DELETE are unimportant commands to predict. Also, numerous observation instances with these values will cause a skew in the data and affect the training negatively. Therefore, any observation with these values for the Event feature was removed from the dataset. Many repeated instances of consecutive observations with identical values for the Event feature were observed in the dataset, causing redundancy, affecting predictions negatively. We removed the repeated commands by comparing the dataset with a shifted copy of it. Any observation that had an identical Event with the shifted copy was dropped.

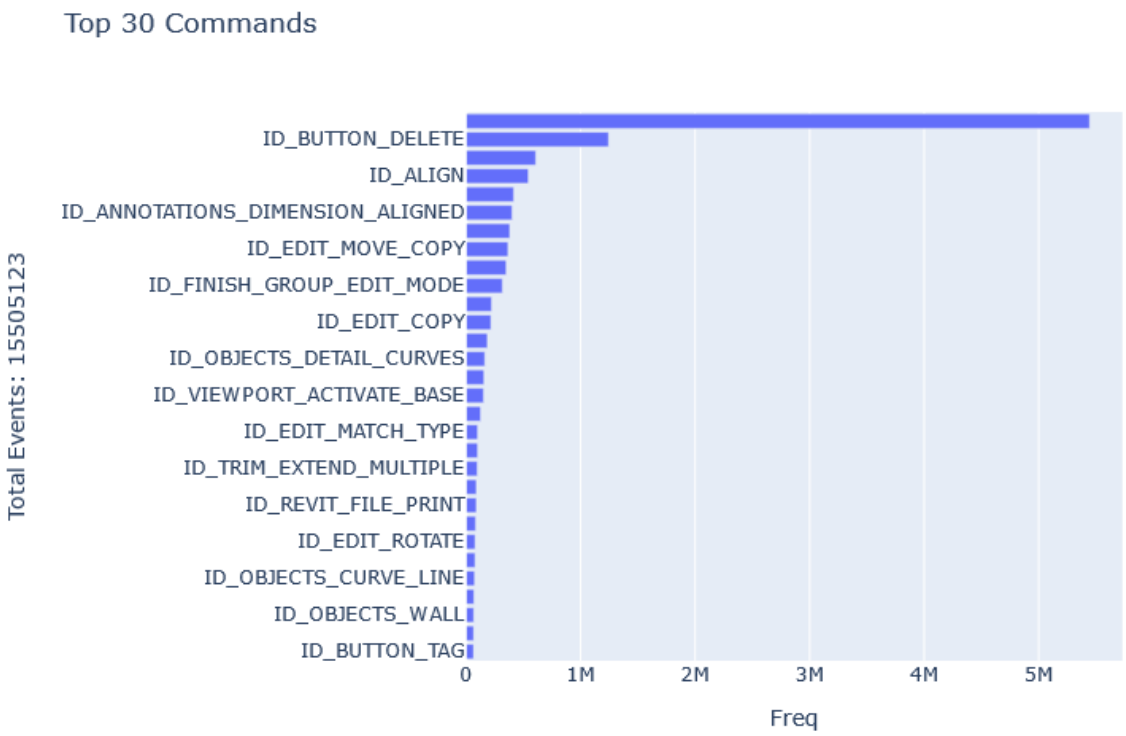

Figure 9: Frequency of Top 30 most frequent commands (excluding cancel)

After we learned about data and their relationships and identified prominent features, we did a preliminary data cleaning and normalization. The next step is to use the data for developing and training the Neural Network with the prepared dataset.

RECURRENT NEURAL NETWORK DEVELOPMENT

Artificial Neural Networks (ANN) are sets of algorithms inspired by our brain cells' inner working. A neural network Takes an input vector, weight parameters, and activation functions and learns to output a vector. The inner cells, called perceptron, are stacked as layers, and data passes through them. Perceptron weights are updated using an optimizer algorithm that minimizes the loss value of each unit. The learning process of the neural network can be categorized as Supervised and unsupervised. In supervised learning, a neural network takes inputs and learns to map them to labels, while an unsupervised learning model learns the relationships and mappings on its own.

Backpropagation and gradient descent are essential concepts in neural network learning. While a supervised model is training, the objective is to adjust weights and parameters in the network to minimize the difference between prediction and true value. This weight adjustment is often made using a backpropagation algorithm, an iterative process in which the network goes through cells in each layer and adjusts their weights to minimize the difference between true value and prediction [27]. An optimization algorithm such as gradient descent adjusts parameters to minimize the loss value calculated by the loss function. It penalizes the network based on the distance between prediction and true value. The loss function is chosen based on the task assigned to the network.[28]

ANNs have gained massive popularity in the last two decades. However, they have been a reality in computer science since 1950, when Nathanial Rochester from the IBM research laboratories led the first efforts to simulate a neural network. Different problems require different ANN types; hence it is essential to understand the problem we aim to solve and select an ANN that is a good fit for the goal.

While in a classical neural network, the inputs are assumed to be independent, for temporal and sequential data, inputs at each time step are dependent on previous inputs. The deep learning model must learn these relations; however, classic ANN cannot capture these temporal relations.

Recurrent Neural Networks (RNN) (Figure 10 source [29]) are a class of ANN that share weights through different time steps, allowing persistence of information from input across the network. Essentially RNN is a sequence of neural networks that perform the same function for every input while information circulates through them. At each time step, in addition to the new input, information from the previous time step will be added; therefore, the produced output is influenced by present and past information.

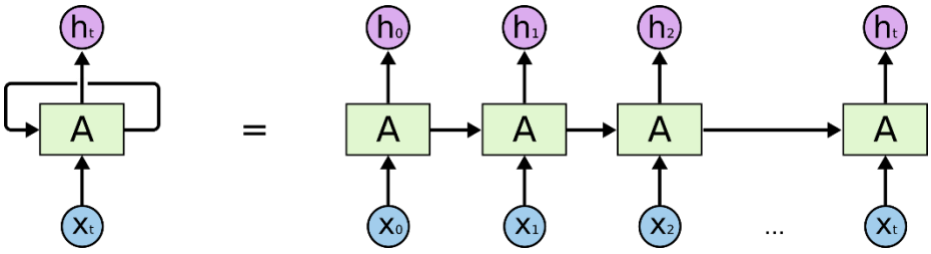

Figure 10: An unrolled RNN showing multiple copies of the same network each passing data to their successor.

RNN leverages internal cell state (memory) to process inputs, making them suitable for tasks involved with sequential data such as language processing, sequence prediction and classification, and time-series predictions. Given the sequential structure of the dataset we created, RNN was a good fit for developing the prediction model. RNN applications can be categorized into five categories based on input and output lengths (Figure 11 source [29])

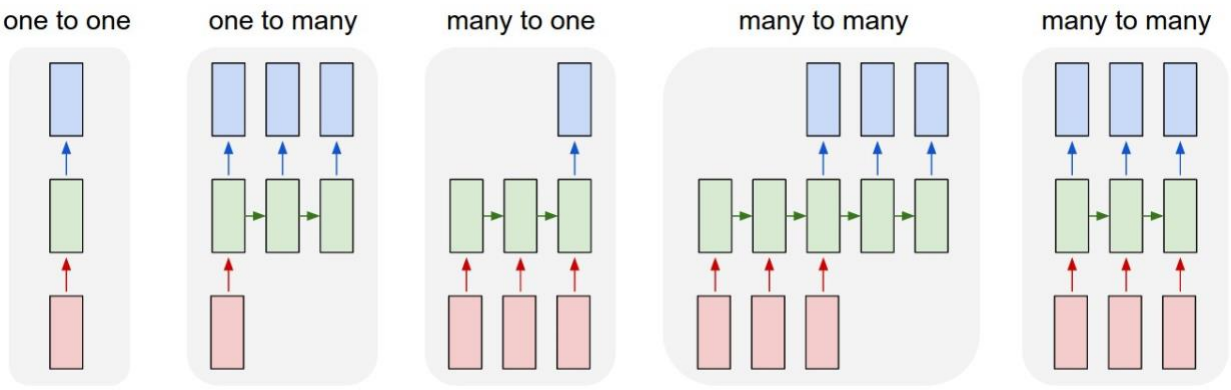

Figure 11: Different models of sequence prediction using RNN

The first type is One to One, which is the same as traditional ANN. The second type, one to Many, is the model that receives a single fixed-size input and outputs a sequence. Image captioning is an example of this type of network that receives a fixed size input (a single vector with image information) and outputs a sequence of data (a sentence describing the image). Many to many models are another type of RNN that takes multiple inputs and gives multiple outputs. Two kinds are based on whether the input and output lengths are equal or not. The latter is used for named entity recognition. When the input is a data sequence and outputs a fixed-size value, the model is referred to as many to one. The model developed for this research is from this category since our inputs are sequences of multiple unique values and model outputs are a single label.[30]

As illustrated in Figure 10, a standard RNN unit that takes input from the previous step and the current output has unit structure. The unit includes an activation function often "tanh" that takes the cell's inputs and remaps it to a value between -1 to 1.

The vanishing gradient problem is a shortcoming of standard RNN, as the network backpropagates through time and calculates the gradient to update the network's weights. If the previous layer affects the current layer by a small amount, then the gradient value will be small

and vice versa. Vanishing gradients will cause the gradients to shrink down exponentially as the network backpropagates. Smaller gradients will fail to update weights effectively; hence, the network does not learn the effect of earlier inputs causing short term memory problems. For predicting user actions from a sequence, it is essential to have the context of the previous task and the recent ones to achieve more accurate results. Therefore, an RNN capable of capturing this long term and short-term memories is essential.[31]

LSTM and GRU

LSTM (Long-Short Term Memory) and GRU (Gated Recurrent Units) are types of RNN developed to address the vanishing gradient and short-term memory problem. The inner mechanism of these units, which is called gate, controls the flow of information. It learns what information in the sequence is important, keeps it, and passes the relevant information to the next unit. This mechanism enables LSTM and GRU to produce a state of results in the tasks that have sequential characteristics. [22], [32]

LSTM, in addition to cell state, has an input, output, and forget gate. Cell state carries the relevant information (long term memory) across the network, and the gates add or remove additional information to it along the way.

Forget gate, which decides whether information should be kept or discarded, uses a sigmoid function that remaps the value between 0 and 1, and values close to zero will be discarded. In contrast, values close to one will be multiplied by the cell state.

The input gate takes the previous hidden state and current input and passes it to a sigmoid function to decide what is important by keeping values close to 1. Hidden state and

current input also get passed to tanh function, and both values from tanh and sigmoid will be multiplied and added to the cell state.

Lastly, the output cell decides what should be passed to the next cell as the hidden its state. The output gate takes the previous hidden state and current input and passes them to a sigmoid function. The result gets multiplied by the tanh of the current state, and the result will be passed as the hidden state of the next cell. Figure 12 (source [29])illustrates the processes of internal gates for both LSTM and GRU.

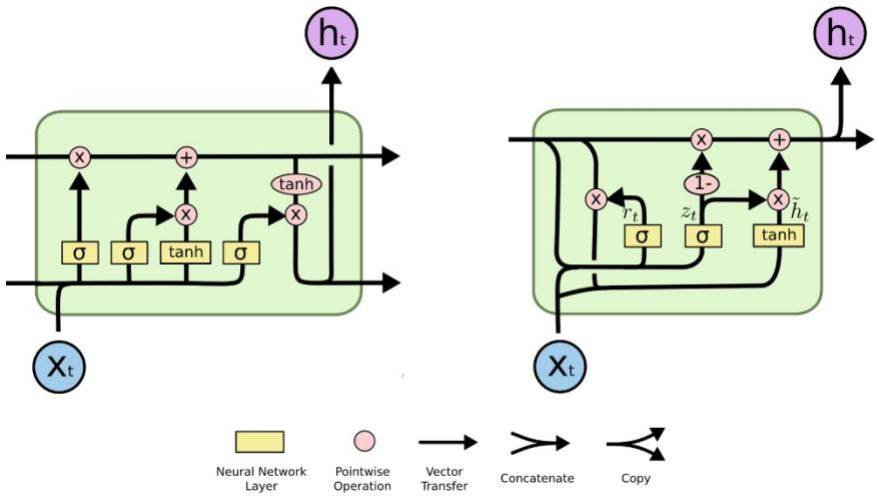

Figure 12: Inner structure of gates for LSTM cell (right) and GRU (left)

GRU is a modern variant of LSTM, which has a similar gate structure. It consists of a reset gate and update gate and only uses the hidden state to pass information from the previous cell. The reset gate controls how much information of the prior step is kept. The update gate is responsible for how much information should be added to information from the last step and carried on to the next step. The simplification of gates results in shorter training time and better model performance. Multiple RNNs with LSTM or GRU cells were developed, trained, and

evaluated to select suitable ones for our task. Throughout the process, unique features, data configurations, and parameters were tested to achieve desired results.

All Machine learning development processes from loading and preparing the data to deployment was done using TensorFlow [33] and Keras [34]. They are open-source deep learning frameworks that facilitate the development of deep learning models by providing various tools and utilities for different development stages, from data pre-processing to deployment. The development of RNN was done in the Google Colab environment, a cloud-based interactive python notebook that provides access to powerful machines equipped with GPUs for machine learning developments.

MODEL EXPERIMENTS

The task formulated to tackle was developing and training RNN that takes a Sequence of Events with a defined length and outputs the command proceeding the sequence. Training and evaluating consists of several steps: mapping data from strings to integers, creating inputs and labels from the sequences, building the neural network with LSTM or GRU units, and finally training the model. After each training model's performance was evaluated by accuracy, loss metrics and predictions were generated to be compared against the actual sequences and then were evaluated by their relevancy to the input sequence.

Event feature values were selected from the refined dataset and used to create input sequences and labels for training. Neural networks cannot be trained on strings; therefore, string sequences must be mapped to a numerical representation. A lookup table was created to map each of the 1,541 unique values in the initial data to an integer value. Since we wanted to interpret the prediction results and integers are not interpretable by humans, we had to convert them back to the actual strings; another lookup table was created to map integer predictions to original strings. For creating input sequences and labels, the chronologically ordered events were split into sequences of length 100. The input sequence length is an important parameter, and different lengths were tested throughout the process. A corresponding label was created for each input sequence with the same length and items shifted one over. Inputs and labels were split into batches of size 64 and then shuffled. The neural network goes through all the inputs within each batch and then backpropagates to update model weights to reduce loss value using a stochastic gradient descent algorithm.

The first model was created with four layers: trainable embedding, dropout, GRU with 1024 cells, and a fully connected dense layer for outputting the prediction. The embedding layer creates N-Dimensional space and turns the integer representation of unique inputs to a vector in the embedding space. Embedding will allow for a better training performance on large sparse inputs like the event sequences data we are using. It will also enable the model to learn the contextual relation between vectors by placing similar inputs closer together in the N-Dimensional embedding space. As the model trains for more epochs, inputs are updated, and their representation improves. Different approaches are used for embedding, especially for natural language data. One-hot encoding, embedding layer, and pre-trained embeddings such as GloVe are among the most used ones. Pre-trained embedding was not an option for this research since pre-trained embeddings are trained on a corpus of natural language texts and cannot represent our data, so we had to use an embedding layer to train ours.

Dropout is used to regularize the model and prevent overfitting; deep neural nets are prone to overfitting, drop out is a technique for addressing this problem. The key idea is to "randomly drop units (along with their connections) from the neural network during training; This prevents units from co-adapting too much." [35]

A final dense layer transforms the logits (LSTM or GRU's numerical output) into probability with a SoftMax activation function. SoftMax takes the exponents of each output and then normalizes them, so their sum equals 1. The output with the highest value will be the prediction for the given input.

The model was compiled, and a Sparse categorical entropy loss function and Adam optimizer with a learning rate of 0.001 were attached. The loss function evaluates the model at

each training step on how much difference exists between the predictions and actual labels for each input. Then loss values are passed to an optimizer often to update the weights and reduce loss in the next epoch.

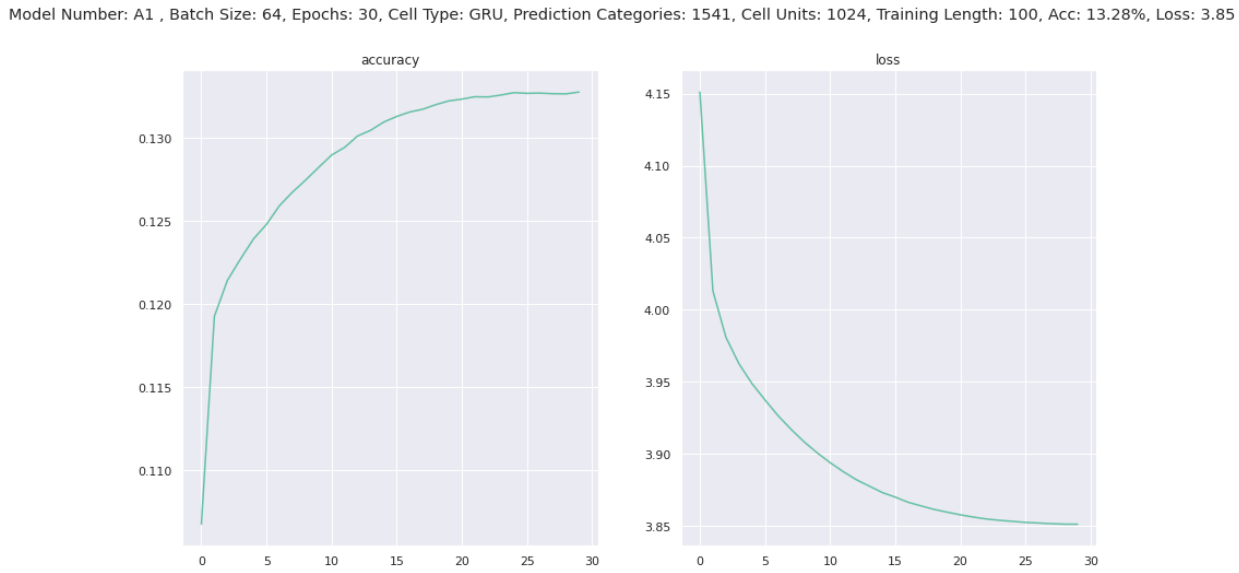

Figure 13: Loss and Accuracy score for the first model

Figure 13 shows the first model's parameters, and loss and accuracy scores at each training step; model prediction accuracy reached 13.2% after 30 epochs. In addition to quantitative metrics, we wanted to evaluate the interpretability of the model's predictions as Revit workflows and compare it with actual commands following the input to check similarities between them.

The defined prediction function takes a seed sequence, trained model, and a diversity parameter and predicts the next event. Since the goal is to predict a sequence of next actions, the seed sequence is combined with the prediction and is fed back to the model to get a new prediction; this combining allows for more context in prediction. This process repeats for as many predictions as needed. The diversity parameter determines how much randomness is added to the predictions. A lower value means a more accurate representation of the actual data;

however, it might cause getting the same output consecutively; hence, different values should be tested to get the best value for each case.

In the sequence generated from 10 items input, predictions and ground truth sequence did not have any similarities in the type of commands; also, it was observed that some commands with higher frequencies such as ID_EDIT_ALIGN were repeated in the generated sequence. In a shorter sample with length five, a similar pattern was seen. ID_EDIT_UNDO, which is the command ID for undoing, was the last item in the input sequence, and might have contributed to the poor predictions since it is a general command. These types of inputs should be removed from the sequences.

For another experiment, the model was trained for more epochs. After 80 epochs of training finished, it reached an accuracy of 13.4%, showing no significant improvement. The generated predictions were also not related logically, and repetition of high frequency commands was still present in the predictions. Next was to test if changing the sequence length can improve the results, so inputs and labels with varying lengths were created. Although longer sequences can give more context to the model for learning and predicting next commands, longer sequences will also increase the probability of including irrelevant commands that do not represent meaningful Revit workflows. After developing and training multiple models with different input lengths using both GRU and LSTM, cells accuracy did not increase beyond 13.8 % and prediction results were still irrelevant to the input sequences.

A large number of categories and lack of enough data for training was potentially the main factor causing the poor performance of different models. With 1541 categories, the neural network fails to learn the relationships between inputs and proper mapping between inputs and

labels. To test if a reduced number of categories and more normalized distribution of events can improve the predictions, we further processed data to identify and discard irrelevant commands or have frequencies less than a threshold defined. We removed observation with Event or Access values with frequencies less than 100; additionally, observations from users and projects with the total number of observations less than 500 were also dropped from the dataset. After data refinements, the number of categories was reduced to 464, and new sets of inputs and labels were created.

An LSTM with 1024 units was built and trained on the dataset with reduced categories for 60 epochs, reaching 12.4% accuracy. However, predictions generated with this model were more relevant. The predicted sequences were similar to the actual commands following the input sequence. Some predictions were observed in the same category of the commands in the input sequences. For instance, ID_FINISH_SKETCH was predicted for an input sequence with ID_FINISH_SWEEP, which both commands from conceptual massing workflow in Revit.

After trying different models and parameters for training and, evaluating them, none of the models achieved a satisfactory performance. Only a slight improvement was seen when cell type was changed to LSTM, and the number of categories was reduced. However, further improvements on the accuracy and interpretability of predictions on test data were essential.

To achieve better results and predictions, the process of input and label creation was revisited. Initially, we ordered all the commands chronologically and split them into sequences of desired lengths. By doing so, we were ignoring the proper order of commands that the user was executing. For instance, some input sequences might have included commands from two

different users or include commands that are a day apart. Training the model on inputs that do not represent proper workflows, unpredicted results, and poor performances was inevitable.

To address the issue with sequence creation and proper representation of workflows, commands by each user for each project were separated. They were grouped into sessions based on UserName, FileName, and the time gap between each consecutive command. Whenever the Username or file name changed or the time gap was more than 10 minutes, a new session was defined; 246,941 sessions were extracted from the dataset using this logic. After loading new data and mapping strings to their numerical representation, labels, and inputs for sequences of length 20 were created, same as the previous experiment length of the sequences were left as a parameter to tune. For creating multiple training examples for each session, first 20 commands were set as input and 21^{st} commands were set as the label then, shifting the 20-item window over the data by one index to get the second item through 21^{st} as input and 22^{nd} as the label, and so on. The 19,535,112 created inputs and labels and were split into a train and test set with a 0.7 - 0.3 ratio. The test set is the data that is not shown to the model during the training process and is used to evaluate its performance and the ability to generalize on unseen data after training.

Multiple LSTM models with different parameters and input sequence lengths were built, trained, and evaluated by their loss and accuracy values as well as their predictions on sample sequences. Layers added to the first network were: an embedding layer with 100-dimensional vector, an LSTM layer with 64 units and a built-in dropout, an intermediate dense layer for adding additional representational capacity to the network, a dropout layer, and for the last layer, a fully connected dense layer with a SoftMax activation function to output predictions by producing the probability for each category. The model trained for 80 epochs after it was compiled with Adam optimizer and a categorical loss function.

The trained network reached 34.4% accuracy, exhibiting that the new method for creating inputs had a positive impact and improved the model's performance compared to the results from the first experiment. To check how the model's performance compares to just random guessing, we calculated accuracy if we were to use the most frequent labels to make predictions and compared it to the trained model's accuracy results. Using ID_ALIGN, the most frequent label model achieved 8.1% accuracy, and by using the top 9 frequent commands and their frequency as probability, predictions were made for all the inputs, and total accuracy was 2.91%. These two experiments demonstrated that the model outperforms both guessing using the most frequent word and using relative word frequencies to make predictions, meaning the network learned from the data to some extent.

Sets of predictions were generated from sample input sequences with lengths of 5, 10, and 20. It can be said that the commands in the predicted sequences were relevant to the input sequence, and they could be interpreted as a continuation of the commands in input to some extent. Other experiments for improving the results were training the model for 150 epochs, building models with 2 LSTM layers and a Bi- Directional LSTM layer, and finally training with input sequence lengths of 10 and 100. By stacking multiple LSTM layers and creating a deeper network, a more complex representation of inputs can be learned. However, better results are not guaranteed. The hidden states generated in the first layer are passed as inputs for the second layer, enabling the model to learn a different input representation. The Bidirectional LSTM layer will allow the model to go over the input's sequences in forward and backward directions, enabling it to learn more about the context of inputs. Finally, different input sequence lengths

can affect the model's training depending on how much context it is getting for predicting the next command.

Model	LSTM/ GRU	Input Length	Categories	Epochs	Loss	Accuracy
B1	LSTM	20	478	80	2.52	34.42%
B 2	LSTM	20	478	120	2.52	34.51%
B 3	2 LSTM	20	478	80	2.50	35.24%
B 4	Bi-Dir LSTM	20	478	80	2.49	34.89%
B 5	LSTM	10	478	80	2.49	34.87%
B 6	LSTM	100	478	80	2.49	34.85%

Figure 14: final model results with various parameters

DISCUSSIONS AND FUTURE WORK

This research aimed to explore potential of Revit Journals as a non-intrusive user interaction data source. Data extracted and structured from Journals were analyzed and visualized to uncover implicit user design and work patterns, for instance what are the most used commands per user or project. This type of insight enables managers and leaders to better understand the project's requirements and team members' capabilities to better plan the project timeline. Additionally, Journal data was used to generate inputs and labels to train a command suggestion supervised machine learning model. Series of different Recurrent Neural Network models were developed and tested to predict next actions following input sequences extracted from journals.

Different challenges had to be dealt with through the process. First challenge was extracting important features and values and normalizing the data since the initial extracted data was imbalanced and had noise. Creating inputs and labels with a large number of categories was another challenging part of RNN development. After taking different approaches in extracting sequences and creating inputs and labels and training model with different hyperparameters, the most significant improvement in results and performance was observed when the label and input creation process was changed and the number of categories was reduced by discarding values observations with less importance. That being said, selecting a model suitable to the task and tuning the hyperparameters is an integral part of the process that cannot be overlooked.

The next steps in further developing this project would be conducting a series of user studies to further evaluate the relevancy, novelty, and usefulness of predictions of the RNN by

having the users respond to the questionnaire. It is important to have input from the users and their interpretation of commands since relying on quantitative metrics is not enough to assess the predictions. For any input sequence there are variety of labels possible, however metrics such as accuracy evaluate the model based on single label for each input. Using the finding from the user study we can further develop the model and fine tune data and parameters.

To develop the user study, it is essential to integrate the model with Revit interface. Preliminary attempts were done for integration but fully integrating was beyond the scope of this research. Revit official API does not expose methods for listening to the command ID, we were able to use UI. FRAMEWORKSERVICES namespace and use getActiveCommandID to listen to commands as they get executed and print Command IDs. PyRevit an open-source rapid development python framework for Revit was used for the interface prototype.

The collected data from logs have great potential to be further studied. For instance, we can identify the sequence of commands that resulted in warnings or errors and train the model to predict and alert the user if the commands they are executing might cause errors. Warnings and errors are common in BIM files. They cause the model to slow down or crash, resulting in losing work or important data. Additionally, the same approach can be taken with the cancel commands. The high frequency of using it by users can be an indicator of how often they activate tools unintentionally and must cancel out from it. Every time this sequence happens, a fraction of time is lost; given the high frequency of this sequence the total lost time can be significant. A recommender system capable of suggesting suitable commands can prevent such lost time and improve the users' working experience.

Development in machine learning and deep learning is growing exponentially, and sophisticated language models such as GPT3 produce extraordinary results, such as texts that cannot easily distinguish whether a real human created it or not, are becoming more available. Journals capture the exact commands and actions user takes to design something in Revit. One can replay a Journal file by simply dragging it to Revit and see each step being reproduced. Suppose resources and data become available to train generative models like GPT-3 on a collection of Journals. It is not far from reality to develop a generative model that generates new Journals that Revit can execute to create a building project. There is exciting potential in exploring this unconventional approach to develop an automated generative system to create new building designs that instead of using parametric and relational approaches generate design by learning design process from actions logs of architects working on various projects.

CONCLUSIONS

This research implements a process to understand Revit Journals to extract and structure data about user interactions with the software. This data has detailed information about the commands that each user has executed for the projects they worked on and any errors raised during the process. By analyzing the Journal data, we can gain insights into user design and work patterns and have a holistic understanding of the design productivity.

A Recurrent Neural Network model a class of Artificial Neural Networks that utilizes internal state to learn from sequential data. We developed a sequential prediction model and trained it on the labels and inputs created from the user interactions data extracted from log files. This model can be integrated with the Revit interface to serve as command recommender system to suggests suitable commands to the user based on their actions. This will enable users to learn novel commands and improve their productivity by reducing errors and decreasing navigation time. This work demonstrates the potential usages of Journals as a data source to learn about how architects design spaces and explores using deep neural nets as s way to leverage this data to assist designers.

REFERENCES

[1] F. H. Abanda, D. Mzyece, A. H. Oti, and M. B. Manjia, "A study of the potential of cloud/mobile BIM for the management of construction projects," *Appl. Syst. Innov.*, vol. 1, no. 2, p. 9, 2018.

[2] S. Azhar, "Building Information Modeling (BIM): Trends, Benefits, Risks, and Challenges for the AEC Industry," *Leadersh. Manag. Eng.*, vol. 11, no. 3, pp. 241–252, Jul. 2011, doi: 10.1061/(ASCE)LM.1943-5630.0000127.

[3] R. Sacks, L. Koskela, B. A. Dave, and R. Owen, "Interaction of lean and building information modeling in construction," *J. Constr. Eng. Manag.*, vol. 136, no. 9, pp. 968–980, 2010.

[4] K. Ullah, I. Lill, and E. Witt, "An Overview of BIM Adoption in the Construction Industry: Benefits and Barriers," in *10th Nordic Conference on Construction Economics and Organization*, vol. 2, I. Lill and E. Witt, Eds. Emerald Publishing Limited, 2019, pp. 297–303.

[5] N. Ham, S. Moon, J.-H. Kim, and J.-J. Kim, "Economic Analysis of Design Errors in BIM-Based High-Rise Construction Projects: Case Study of Haeundae L Project," *J. Constr. Eng. Manag.*, vol. 144, no. 6, p. 05018006, Jun. 2018, doi: 10.1061/(ASCE)CO.1943-7862.0001498.

[6] H. W. Lee, H. Oh, Y. Kim, and K. Choi, "Quantitative analysis of warnings in building information modeling (BIM)," *Autom. Constr.*, vol. 51, pp. 23–31, Mar. 2015, doi: 10.1016/j.autcon.2014.12.007.

[7] R. Fulford and C. Standing, "Construction industry productivity and the potential for collaborative practice," *Int. J. Proj. Manag.*, vol. 32, pp. 315–326, Feb. 2014, doi: 10.1016/j.ijproman.2013.05.007.

[8] L. Zhang, M. Wen, and B. Ashuri, "BIM Log Mining: Measuring Design Productivity," *J. Comput. Civ. Eng.*, vol. 32, no. 1, p. 04017071, Jan. 2018, doi: 10.1061/(ASCE)CP.1943-5487.0000721.

[9] N. Khean, A. Fabbri, D. Gerber, and M. H. Haeusler, "Examining Potential Socio-economic Factors that Affect Machine Learning Research in the AEC Industry," in *Computer-Aided Architectural Design. "Hello, Culture,"* Singapore, 2019, pp. 247–263, doi: 10.1007/978-981-13-8410-3_18.

[10] M. Guzdial, "Deriving Software Usage Patterns from Log Files," Georgia Institute of Technology, Technical Report, 1993. Accessed: Apr. 03, 2021. [Online]. Available: https://smartech.gatech.edu/handle/1853/3650.

[11] R. Das and I. Turkoglu, "Creating meaningful data from web logs for improving the impressiveness of a website by using path analysis method," *Expert Syst. Appl.*, vol. 36, no. 3, Part 2, pp. 6635–6644, Apr. 2009, doi: 10.1016/j.eswa.2008.08.067.

[12] D. Peacock, *Statistics, Structures & Satisfied Customers: Using Web Log Data to Improve Site Performance*. For full text: http://www, 2002.

[13] D. Li, G. Zhao, Z. Wang, W. Ma, and Y. Liu, "A Method of Purchase Prediction Based on User Behavior Log," in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, Nov. 2015, pp. 1031–1039, doi: 10.1109/ICDMW.2015.179.

[14] P. Lopes and B. Roy, "Dynamic recommendation system using web usage mining for e-commerce users," *Procedia Comput. Sci.*, vol. 45, pp. 60–69, 2015.

[15] S. Dumais, R. Jeffries, D. M. Russell, D. Tang, and J. Teevan, "Understanding User Behavior Through Log Data and Analysis," in *Ways of Knowing in HCI*, J. S. Olson and W. A. Kellogg, Eds. New York, NY: Springer, 2014, pp. 349–372.

[16] E. Adar, J. Teevan, and S. T. Dumais, "Large scale analysis of web revisitation patterns," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, Apr. 2008, pp. 1197–1206, doi: 10.1145/1357054.1357241.

[17] N. Zhang, Y. Tian, and M. Al-Hussein, "A Case Study for 3D Modeling Process Analysis based on BIM Log File," *Modul. Offsite Constr. MOC Summit Proc.*, pp. 309–313, May 2019, doi: 10.29173/mocs108.

[18] S. Yarmohammadi, R. Pourabolghasem, and D. Castro-Lacouture, "Mining implicit 3D modeling patterns from unstructured temporal BIM log text data," *Autom. Constr.*, vol. 81, pp. 17–24, Sep. 2017, doi: 10.1016/j.autcon.2017.04.012.

[19] Y. Pan and L. Zhang, "BIM log mining: Learning and predicting design commands," *Autom. Constr.*, vol. 112, p. 103107, Apr. 2020, doi: 10.1016/j.autcon.2020.103107.

[20] L. Lü, M. Medo, C. H. Yeung, Y.-C. Zhang, Z.-K. Zhang, and T. Zhou, "Recommender systems," *Phys. Rep.*, vol. 519, no. 1, pp. 1–49, Oct. 2012, doi: 10.1016/j.physrep.2012.02.006.

[21] Y. Zhang *et al.*, "Sequential click prediction for sponsored search with recurrent neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2014, vol. 28, no. 1.

[22] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” *ArXiv14123555 Cs*, Dec. 2014, Accessed: Feb. 01, 2021. [Online]. Available: http://arxiv.org/abs/1412.3555.
[23] H. Soh, S. Sanner, M. White, and G. Jamieson, “Deep sequential recommendation for personalized adaptive user interfaces,” in *Proceedings of the 22nd international conference on intelligent user interfaces*, 2017, pp. 589–593.
[24] M. Wiebe, D. Y. Geiskkovitch, and A. Bunt, “Exploring User Attitudes Towards Different Approaches to Command Recommendation in Feature-Rich Software,” in *Proceedings of the 21st International Conference on Intelligent User Interfaces*, New York, NY, USA, Mar. 2016, pp. 43–47, doi: 10.1145/2856767.2856814.
[25] J. Matejka, W. Li, T. Grossman, and G. Fitzmaurice, “CommunityCommands: command recommendations for software applications,” in *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, New York, NY, USA, Oct. 2009, pp. 193–202, doi: 10.1145/1622176.1622214.
[26] M. Bastian, S. Heymann, and M. Jacomy, “Gephi: an open source software for exploring and manipulating networks,” in *Proceedings of the International AAAI Conference on Web and Social Media*, 2009, vol. 3, no. 1.
[27] P. J. Werbos, “Backpropagation through time: what it does and how to do it,” *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.
[28] S. Ruder, “An overview of gradient descent optimization algorithms,” *ArXiv Prepr. ArXiv160904747*, 2016.
[29] “Understanding LSTM Networks -- colah’s blog.” https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed Apr. 09, 2021).
[30] “The Unreasonable Effectiveness of Recurrent Neural Networks.” https://karpathy.github.io/2015/05/21/rnn-effectiveness/ (accessed Apr. 20, 2021).
[31] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 6, no. 02, pp. 107–116, 1998.
[32] F. A. Gers, J. Schmidhuber, and F. Cummins, “Learning to forget: Continual prediction with LSTM,” 1999.
[33] Martín Abadi *et al.*, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015.
[34] F. Chollet and et al., “Keras,” 2015. https://github.com/fchollet/keras.
[35] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.