Department of Computer Science

MSc Artificial Intelligence

Academic Year 2023-2024

Enhancing Self-Attention Models for Link Prediction in Dynamic Graphs

Bruno Fenali Almeida

Supervisor: Weibo Liu

A report submitted in partial fulfilment of the requirement for the degree of Master of Science

Brunel University
Department of Computer Science
Uxbridge, Middlesex UB8 3PH
United Kingdom
Tel: +44 (0) 1895 203397
Fax: +44 (0) 1895 25168

Abstract

This dissertation investigates the influence of architectural modifications on self-attention mechanisms for link prediction tasks in dynamic graphs. Graphs are a versatile data structure and are used in various domains, from social networks (Fan et al., 2019) to molecule simulations (Gilmer et al., 2017). The usage of self-attention mechanisms in dynamic graphs has shown promising results, as demonstrated by the Temporal Graph Benchmark (TGB) (*Leaderboard TGB*, 2024). Nevertheless, some models perform subpar, even when using self-attention. Therefore, comprehending the impact of architectural changes enables the development of more robust and expressive models.

A quantitative approach was taken, applying established practices from static graphs to dynamic graph models. The research focused on four distinct architectures, with a baseline model and three novel strategies: Cumulative Message Passing (CMP), Dual Embedding, and Temporal Walk Embedding approach. These models were iteratively trained, evaluated, and compared using two TGB datasets.

The findings show the effectiveness of the CMP architecture, which enhances node embeddings by aggregating temporal information. This method has outperformed the baseline model and demonstrated neighbourhood information's importance in dynamic graphs. The Dual Embedding strategy, designed to differentiate source and destination nodes, shows potential, particularly in enhancing CMP. Temporal Walks, although the weakest performer, suggest the possibility for further exploration.

This research contributes to the field by experimenting different architectures effect on self-attention mechanism. Future work could explore broader datasets, alternative sequence definitions, and new approaches to time embedding and self-attention.

Contents

1 Introduction

Graphs are a versatile data structure used to model relationships in a wide range of subjects, from social networks (Fan et al., 2019) and recommendation systems (Wang, Zhao, et al., 2023) to more specialised areas like physical environment modelling (Pfaff et al., 2021) and molecule simulations (Gilmer et al., 2017). When combined with deep learning, graph-based models have proven especially powerful in capturing complex patterns in static and dynamic environments.

A fundamental distinction in graph modelling is between Static and Dynamic graphs. Static graphs have a fixed topology, making them suitable for tasks like node classification and graph-level predictions (S. Wu et al., 2022). In contrast, Dynamic graphs capture changes over time, aligning more closely with real-world scenarios where relationships between entities evolve, such as in temporal social networks. Dynamic graphs are particularly suited to downstream tasks aiming to predict the existence of an edge at a particular time (link prediction) (S. Huang et al., 2023).

The recent use of self-attention mechanisms has significantly impacted both static and dynamic graph models. Self-attention, which has been consistently pushing the boundaries in Natural Language Processes (Wolf et al., 2020)(Vaswani et al., 2017) and Computer Vision (Dosovitskiy et al., 2021)(Carion et al., 2020), is now being applied to dynamic graphs with promising results. In 2023, a new benchmark called the Temporal Graph Benchmark (TGB) highlighted the dominance of attention-based methods in link prediction tasks, with four out of five datasets showing attention-based models leading in performance (S. Huang et al., 2023). Each with its own unique architecture, these models showcase a range of different performances across the datasets. While some are at the top of the leaderboard, others only perform better than heuristic-based methods. The significant variation in the performance of these models suggests that specific architectural choices can heavily influence outcomes.

In common, all these works have focused on developing novel architectures that leverage self-attention for dynamic graphs, but there is a notable gap in studies that systematically compare how different architectural modifications impact model performance. Understanding which aspects of these architectures contribute most to the performance of the self-attention mechanism is crucial for designing more robust and efficient models.

1.1 Research Aim and Objectives

Motivated by the findings on the varying performance of dynamic graph models with attention mechanisms and the insufficiency of research exploring how different changes impact the performance of the self-attention models, this research aims to answer the question:

How do specific architectural modifications in dynamic graph models influence the effectiveness of self-attention mechanisms in link prediction tasks?

The main objectives are:

- To explore static graph concepts and adapt them to dynamic graph models.
- To explore how to encode the graph topology information to be used by the self-attention model.
- To evaluate how the designed architectures enhance self-attention models for dynamic graph link prediction using datasets from the Temporal Graph Benchmark.
- To derive practical insights that can guide the optimal application of self-attention mechanisms in real-world dynamic graph learning tasks.

The scope does not include exploring different architectures for the sequence creation or investigating the different layers for multi-head attention and different types of attention mechanisms.

This work will contribute to the literature by exploring elements that improve or detract from the performance of models using self-attention, offering insights that could lead to the development of more robust models. Additionally, as Artificial Intelligence (AI) models grow in complexity, there is an urgent need to understand which architectural features are genuinely beneficial, especially in the context of graph-based data applications, which are computationally intense.

A few limitations of this research are the scope, as mentioned before, and the fact that it does not include architectures that change the sequence definition, different types of attention, or different layers in the model. Also, due to computational limitations, the models are only tested in two out of five of the benchmark datasets.

The rest of the work is organised into six main chapters:

- The "Literature Review" presents the current SOTA models and the research that has been the base of the models proposed in the dissertation.
- The "Methodology" justifies the research approach and details the architecture designed to address the research topic.
- The "Experiments" section covers the datasets and the tools and technologies used. It also describes the steps of data analysis, data pre-processing, and model evaluation.
- The "Results" summarises the findings of the model exploration, showing how the evaluation metric performed with each model design.
- The "Discussions" chapter interprets the results and discusses the architectures' impact on the attention model. It also covers the limitations of this work.

- The "Conclusion" briefly reviews the outcomes of the dissertation and suggests future works.

2 Literature Review

Graphs are mathematical structures consisting of two finite sets: the nodes set and the edges set (Harris et al., 2008). Graphs have been classified into different types based on how their nodes connect. Undirected graphs have edges unordered with unordered pairs of nodes, while directed graphs present edges that are ordered pairs. Additionally, graphs can differ in complexity, where multigraphs allow many edges for the same pair of nodes, and weighted graphs contain edges that carry weights on the nodes' connection. In machine learning, the representation of graph data is extremely important for the success of graph-based models (Hamilton, 2020b).

Machine learning models heavily depend on the representation of the data (Goodfellow et al., 2017b). Therefore, feature engineering and feature selection techniques are often applied to the data to help models achieve better results (Russell and Norvig, 2022). Conversely, the concept of Representation Learning assigns the task of discovering the most effective representation of raw data to a machine learning model rather than to a human being, then, the mode is introduced into a pipeline that aims to solve a downstream task (Goodfellow et al., 2017b).

Embeddings are a typical subproduct of a model that employs representation learning. Embeddings are low-dimensional dense vectors, and their space is a manifold derived from the original raw data space (Goodfellow et al., 2017a)(Guillemin and Pollack, 1974). Some authors also define embedding spaces as the latent spaces (Sankar et al., 2020)(Zhu et al., 2016). In this thesis, both definitions are interchangeable. Embeddings are particularly helpful for two main reasons: they are lighter than the original data (low-dimensional dense vectors), and they are sensitive only to changes in the space defined by the manifold, which means that irrelevant or less informative variations orthogonal to the manifold are not captured (Goodfellow et al., 2017a), serving as a filter to the information represented.

The deep-learning literature related to graph problems focuses mainly on representation learning, particularly on how deep-learning models can be used to generate meaningful embeddings for the graph nodes, considering the graph's topology. For a better understanding, one can divide the research field into two subgroups: models related to Static Graphs (Perozzi et al., 2014)(Kipf and Welling, 2017)(Veličković et al., 2018) and related to Dynamic Graphs (Sankar et al., 2020)(Cong et al., 2023)(L. Yu et al., 2023).

2.1 Static Graphs

Graph Neural Networks (GNNs) are a concept that has changed considerably over the years. This section starts with the first idea of GNNs, then explains a modern concept known as Message Passing Neural Networks, describes Graph Convolution Networks and Graph Attention Networks, and finishes with Random Walk and Asymmetric approaches.

2.1.1 Graph Neural Network - GNN

GNNs have been first defined by Scarselli et al. (2009) as a structure similar to Recurrent Neural Networks (RNNs), but its definition evolved with the years. GNN initial idea was to optimise two functions at once via gradient descent: f and g. f is a contraction map representing the connections between the nodes, and g is a non-linear function that generates the encodings of the nodes (Scarselli et al., 2009).

The function f is designed to be a contraction map to ensure that the connection between two nodes maps to a single fixed point in the hyperspace, invariant to the function; thus, by using the Banach Contraction Principle (Goebel and Kirk, 1990), this fixed point is guaranteed to exist and is unique, ensuring convergence (Scarselli et al., 2009).

Unfortunately, the condition of contraction maps and the learning process based on searching for a stable middle point between each pair of nodes has imposed a considerable bottleneck on GNNs (Y. Li et al., 2017). Over the years, modern approaches in deep learning have evolved GNNs, enhancing their expressivity and enabling them to handle larger, more diverse types of graphs (Z. Wu et al., 2021).

2.1.2 Message Passing Neural Network - MPNN

A subclass of modern GNN algorithms named as ConvGNNs aims to propagate information over the nodes through the edges (Z. Wu et al., 2021). The idea have been initially designed by Gilmer et al. (2017) when the authors had identified that a considerable number of models laid in this particular framework called Message Passing Neural Network (MPNN). MPNN is composed of two phases: Message Passing Phase and Readout Phase. The message passing phase runs for a fixed number of steps and propagates the information of the nodes to its neighbours by defining two functions: the message function M_s (1) and the update function U_s (2). Gilmer et al. (2017) define time steps with the letter t, but to avoid any confusion with future topics related to dynamic graphs the letter s will be used instead.

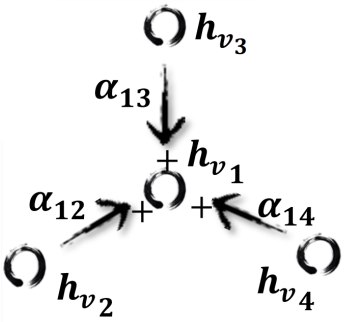


Figure 1: Message being propagated through nodes in a graph (Z. Wu et al., 2021)

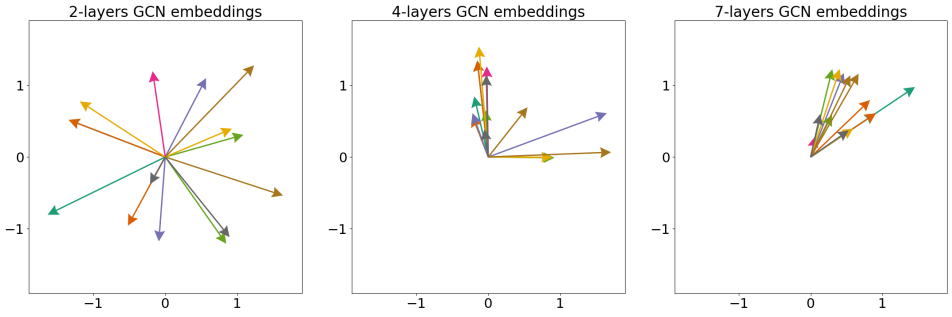
Gilmer et al. (2017) define M_s as the function which builds the representation of the relation between two nodes, u and v. The function considers the embed-

ding of a node u at step s as h_x^s together with the information about the edge that connects both nodes as e_{uv}. After summing all the messages from the neighbours, the representation of the node u at step $s+1$ is updated using the function U_s and its previous representation h_u^s. Considering $N(u)$ as the function that returns a set with all the neighbours of node u, the representation of s after one step of message passing is defined as:

$$m_u^{s+1} = \sum_{w \in N(u)} M_s(h_u^s, h_w^s, e_{uw}) \tag{1}$$

$$h_u^{s+1} = U_s(h_u^s, m_u^{s+1}). \tag{2}$$

The information being propagated on MPNN is the embedding defined on each node. Thus, by aggregating information about its neighbours, each node gains awareness about the whole network based on its topology (Gilmer et al., 2017). The fixed number of times that message passing runs, i.e., the steps, controls how far the message is spread. Each application propagates the information of a further point: the first application aggregates the information 1-hop away from the nodes, the second aggregates the information from the first application with the information from 2-hops away of the node, and so forth (Gilmer et al., 2017). The steps are the number of layers of MPNN algorithms defined on the network. One might notice that the continuous application of many message-passing phases without any counter structure leads to an issue called Oversmoothing, where the embeddings of all the nodes are blended into a similar structure, leading to a subpar expressivity of the model (Bodnar et al., 2022).



(a) 2-layers GCN network (b) 4-layers GCN network (c) 7-layers GCN network

Figure 2: Oversmoothing phenomenon - 2D embeddings created on a dummy graph data

2.1.3 Graph Convolution Networks - GCN

Graph Convolution Networks (GCN) (Kipf and Welling, 2017) are the exponent of ConvGNNs. GCNs have extended the concept of Convolution Neural Networks (CNN) to graphs, a non-euclidean space, by formulating a scalable approximation

of the concept of spectral graph convolution using neural networks (Bronstein et al., 2017). A spectral graph convolution ($\star$) is the convolution of a filter g_θ onto a graph signal x that has been transformed to a spectral domain by using the graph Fourier transformation, which in turn is based on the eigendecomposition of the Laplacian matrix. The spectral graph convolution is defined as

$$g_\theta \star x = U G_\theta U^T x. \tag{3}$$

The matrix U is defined using the eigenvectors of the normalised version of the Laplacian matrix. The Laplacian matrix is defined using the Adjacency Matrix A of a graph and a Degree matrix of its nodes D (Hamilton, 2020c).

GCN approximates the spectral convolution as follows:

$$g_\theta \star x \approx \tilde{D}^{\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta, \tag{4}$$

$\tilde{A}$ and $\tilde{D}$ are the result of normalisation tricks to mitigate the issues with exploding and vanishing gradient descents (Kipf and Welling, 2017). $\tilde{A}$ is defined as an alternative adjacency matrix being equal to $A + I$, being A the adjacency matrix and I the identity matrix, and $\tilde{D}$ the new degree matrix of this alternative adjacency matrix created. Finally, Θ is the learnable parameters of a neural network linear layer, followed by some non-linear function (Kipf and Welling, 2017).

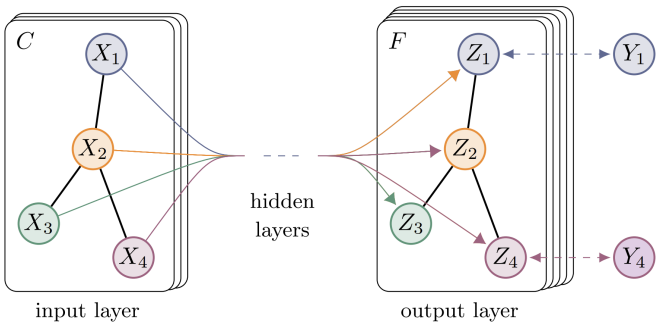


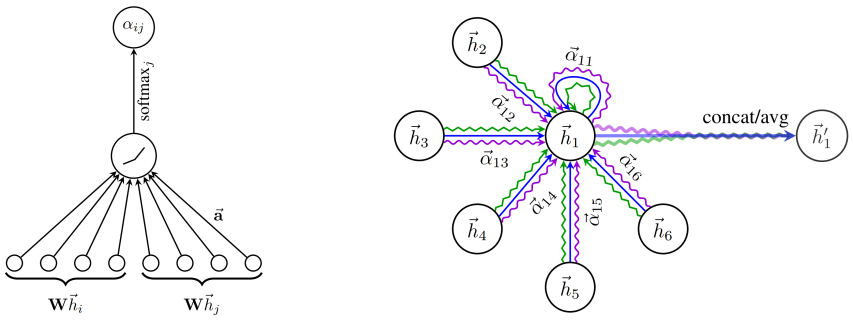
Figure 3: GCN computing a representation for each graph node using the entire graph as input (Kipf and Welling, 2017)

In simpler terms, the approximation that has been proposed by Graph Convolutional Networks (GCNs) introduces self-connections at each node when summing the identity matrix onto the adjacency matrix. The result of the matrix multiplication can be understood as a message passing from the MPNN framework, where the embedding from each node's neighbours, along with its own, is aggregated and then normalised by the square root of the degrees of the source and destination nodes (Hamilton, 2020a).

2.1.4 Graph Attention Networks - GAT

Graph Attention Networks (GATs) enhance the idea of GCN. Instead of normalising the message considering the degree of the nodes, it introduces an attention

mechanism where a factor between 0 and 1 is learned to each edge in order to give different weights for the message of the neighbours of a node (Veličković et al., 2018). Such weights are defined by a unique linear neural network layer followed by a LeakyReLU non-linear function (figure 4a).



(a) one layer attention mechanism (b) multihead diagram, showing 3 heads being used

Figure 4: Diagrams from GAT explaining how attention is calculated and propagated on multihead (Veličković et al., 2018)

Moreover, the weights of the edges of a node are normalised through a softmax so that they sum to one (Veličković et al., 2018). Lastly, the Veličković et al. (2018) has borrowed the multi-head attention concept from Grover and Leskovec (2016) and has implemented its version for GATs. Instead of calculating only one weight for each edge, k parallel weights are calculated, being k the number of heads, and as a result, k aggregated messages are computed. In the last step, those messages are averaged (figure 4b), leading to more stable results than the single head.

2.1.5 Random Walk

Another class of deep learning models is based on graph walks and is more connected to capturing the graph's structure regardless of the usage in some specific downstream task. With an unsupervised learning approach, those methods leverage the concept of random walks to create embedding that solely represents the graph's topology - e.g. DeepWalk (Perozzi et al., 2014) and Node2Vec (Grover and Leskovec, 2016).

Random walks are walks that follow some heuristic that uses random functions - by starting in a node u, a walk is defined by a sequence of connected nodes until it arrives in some destination v (Harris et al., 2008) (Figure 5). DeepWalk has extended the SkipGram algorithm (Mikolov et al., 2013) from language models to the domain of graphs.

DeepWalk aims to capture the conditional probability of the occurrence of other nodes in the walks by performing an arbitrary number of random walks started in each node. It optimises a hierarchical softmax function, which serves as the loss function of a classifier that aims to predict the distribution of the neighbouring nodes (Perozzi et al., 2014). The heuristic used by DeepWalk to build the

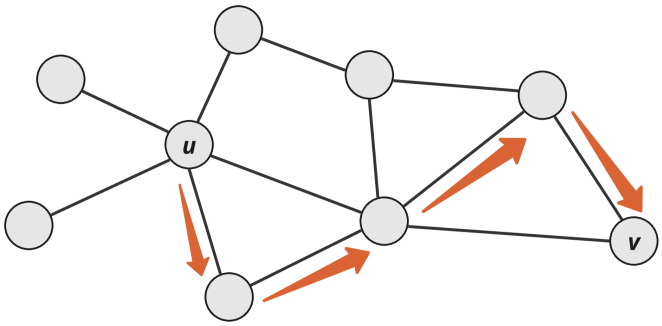


Figure 5: Random Walk from node u to node v

random walk is based on choosing uniformly random a neighbour of the last node of the walk. Node2vec has extended DeepWalk by inserting two new hyperparameters, q and p, which sets the random walk to have a more Breadth-First Search approach by stimulating the walk to go to unvisited neighbours of nodes previously visited, or to have a more Depth-First Search approach, going deeply in the path already started (Grover and Leskovec, 2016). Both methods have the same outcome: an embedding for each node reflecting their proximity among the other nodes on the graph.

2.1.6 Asymmetric Embeddings

Lastly, some models treat the embedding as a matrix, and the final structure is defined by optimising an objective function that measures the relation between the representation of the nodes in this matrix. For instance, HOPE builds the embedding matrix based on a high-order graph proximity metric (such as the Katz Index or Common Neighbors), relying on an approximation of those metrics (Ou et al., 2016). The approximation of the metrics uses Singular Value Decomposition to reduce computational complexity by avoiding the explicit calculation of every pair of nodes (Ou et al., 2016). HOPE then minimizes the difference between the dot product of the node's representations and the approximated proximity metric obtained. Moreover, HOPE is based on directed graphs, and it takes into account asymmetric transitions on them, building two different representations for the nodes, one when the nodes are the source of the edges and another when they are the destination (figure 6) (Ou et al., 2016).

2.2 Sequence

Before introducing algorithms related to dynamic graphs, one must visit and understand Sequence models once they are heavily linked. Sequence models are based on Recurrent Neural Networks (RNNs), and they differ from standard neural networks in the way they operate over a sequence of items. By carrying out a hidden state in each iteration, the network can predict the next hidden state of an item by

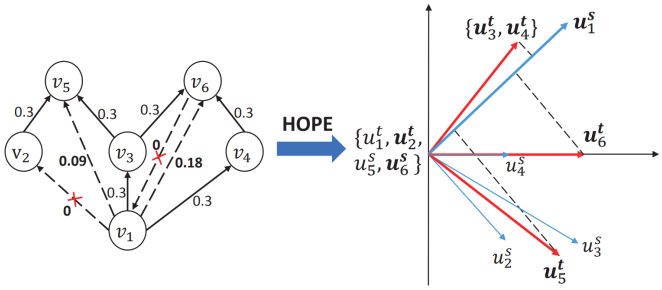


Figure 6: HOPE - distinct node's representations for source and destination on Directed Graphs (Ou et al., 2016)

combining the current item information with the cumulative state predicted on the previous sequence items (Cho et al., 2014).

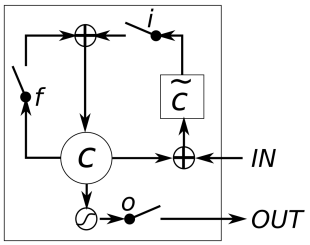
2.2.1 Long Short-Term Memory - LSTM

Long Short-Term Memory (LSTM) is one of the most traditional yet most valuable algorithms in the field of RNNs, entirely based on the concepts of gates and an internal structure called a memory cell (Hochreiter and Schmidhuber, 1997). Originally, LSTM has been defined using two gates, the input and the output gate. The gates are responsible for controlling the information of the sequence that is passed through the memory cell. The input gate controls what is saved on the memory cell. Meanwhile, the output gates control the information the cell emits (Hochreiter and Schmidhuber, 1997).

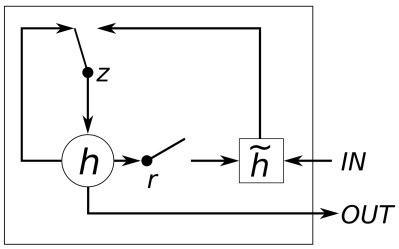
By the work of Gers et al. (1999), LSTMs have been extended with a third gate called the forget gate, enabling the model to discard outdated information (figure 7a). LSTMs manage the flow of information to and from the memory cell during forward and backpropagation by applying different weights and biases to a sigmoid function within each gate (Hochreiter and Schmidhuber, 1997). The gates combined with the memory cell structure are pivotal to making LSTM capable of working with longer sequences without suffering from exploding or vanishing gradient descent (Hochreiter and Schmidhuber, 1997).

2.2.2 Gated Recurrent Units - GRU

Gated Recurrent Units (GRUs) are based on a simplified version of the gate structures proposed by LSTMs. GRUs do not rely on the memory cell concept and have only two gates: the update gate and the reset gate (Cho et al., 2014). The update gate combines and simplifies the input and output gates, whereas the reset gates play a similar role to the forget gate. Since GRUs do not have a separate memory cell, both gates directly manage the previous hidden state passed as input from the last item in the sequence (Gers et al., 1999) (figure 7b). Despite GRUs being



(a) LSTM module diagram: i (input), o (output), and f (forget) are the gates. c and $\tilde{c}$ the memory cell.



(b) GRU module diagram: z (update) and r (reset) are the gates. h is the activation and $\tilde{h}$ the candidate activation.

Figure 7: Diagrams comparing LSTM with GRU (Chung et al., 2014)

significantly simpler than LSTMs, they present similar expressivity, making both viable options for most sequence problems (Chung et al., 2014).

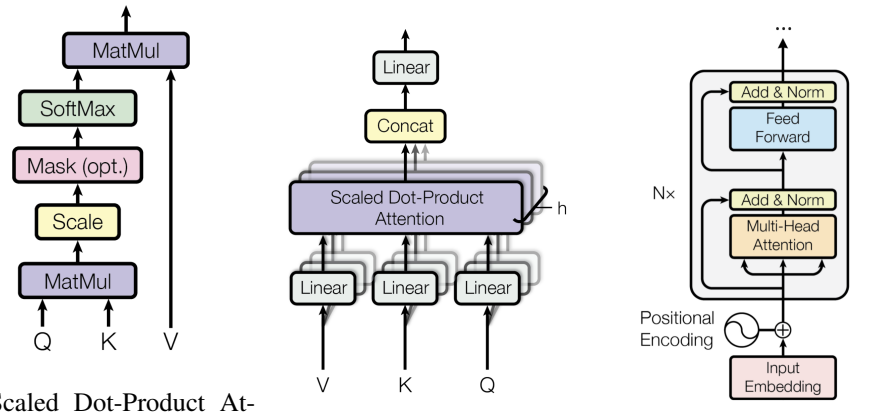
2.2.3 Transformer

Finally, there is the Transformer, which arrives as an alternative to the methods mentioned previously. Similar to the methods mentioned above, an outcome for each sequence item is calculated, but unlike RNNs, Transformer is more parallelised. For all items in the sequence, all the subsequences that generate those items will be considered at once. By doing so, the Transformer heavily parallelises the training and outperforms RNN models that, by their sequential nature, impose a slower learnable process (Vaswani et al., 2017).

The Transformer is built on three fundamental concepts: Scaled Dot-Product Attention (used as a self-attention mechanism), Multi-head Attention, and positional encoding (Vaswani et al., 2017). It is important to pinpoint that, although the Transformer is fundamentally a language model, its ideas have spread around a variety of other deep-learning fields, and the three concepts formerly mentioned are vital to understanding the current state-of-the-art (SOTA) models for dynamic graphs.

The method that the Transformer has to update the inner state of an item in a sequence is by using the information of all the other items. A self-attention mechanism filters all the information and adjusts the influence of the other items onto a given one. The self-attention mechanism which has been developed by Vaswani et al. (2017) is called Scaled Dot-Product Attention (figure 8a). Scaled Dot-Product Attention is a function of three arguments: the queries (Q), the keys (K), and values (V), each one being a representation of each item on the sequence (in many applications, they end up being the same source). A dot product between K and Q is computed, then scaled using the dimension of K, and the result passes through a softmax,

$$ScaledDotProductAttention(Q,K,V) = softmax\left(\frac{Q \cdot K^T}{\sqrt{d_K}}\right) * V. \qquad (5)$$



(a) Scaled Dot-Product Attention

(b) Multi-Head Attention

(c) Transformer Module

Figure 8: From left to right, a bottom-up visualisation of the Transformer (Vaswani et al., 2017)

The softmax converts the attention results into a probability distribution, guaranteeing that the sum of the score is one, allowing a weighted combination of the values. The dot product between K and Q gives a numerical value of the alignment between them. This value adjusts the influence of the values V from all other items in the sequence on the representation of each item (Vaswani et al., 2017).

Vaswani et al. (2017) have also introduced the concept of multihead attention, where multiple attention scores are calculated, then concatenated and passed through a linear layer that aims to combine them to have a final representation (figure 8b).

Lastly, a positional encode is added to each embedding, helping the model to identify the item based on its position in the sequence (figure 8c) (Vaswani et al., 2017). For example, an item that occurs twice in a sequence might have its inner embedding representation defined equally for both occurrences (since it is the same item), but its position encode embedding will vary since this part depends on the position (Gehring et al., 2017). Many other research works also translate the representation of an item in a sequence using the embedding that encodes its space-time position. By doing so, non-sequential approaches can be used even on a sequential problem (L. Yu et al., 2023)(Xu et al., 2020)(Cong et al., 2023)(Gehring et al., 2017).

2.3 Dynamic Graphs

All the graph-related methods discussed above assume that the graphs are static, so information must be obtained from a fixed structure. Nevertheless, real-world applications face a different situation where a mutable graph structure is fronted, and it is expected that the model deals with such mutable networks (B. Yu et al., 2018)(Z. Huang et al., 2020)(X. Li et al., 2020)(Song et al., 2019). Such graphs

are known as Dynamic Graphs.

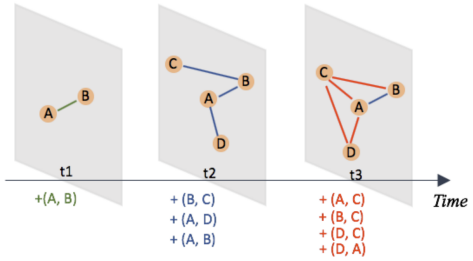


Figure 9: Dynamic Graph changing over time with the addition of new edges (Xu et al., 2020)

Although there are differences among the models that will be presented, one must notice that they share a common goal: dynamic graph methods aim to create an embedding of the nodes such that it is sensitive to space (as on a static graph) while also sensitive to time. Most of the literature uses such representation to run edge predictions in dynamic graphs as the primary way to evaluate their novelty (S. Huang et al., 2023).

Models based on some attention mechanisms are typical, and several will be presented in the following subsections.

2.3.1 Dynamic Self-Attention Network

Dynamic Self-Attention Network (DySat) discretises the time in order to create snapshots of the graph (Sankar et al., 2020). On each snapshot, a variant of GAT, named the Structural Attention Layer, is applied, and a structural static representation is built for that snapshot. The static representations from different timesteps are then evaluated by a temporal self-attention layer inspired by Scaled Dot-Product from Transformer. Nonetheless, as with other discrete methods, DySat falls short of encoding structural information over long periods, and the aggregation of the timestamps in large datasets becomes another hyperparameter to be controlled (Trivedi et al., 2019).

2.3.2 DyRep

Aiming to fix the limitations from discrete methods, Trivedi et al. (2019) presents DyRep, a model that operates continuously by updating the neighbourhood of the nodes every timestamp t that an event has occurred. At time $t + 1$, when a new edge arrives, it updates the embeddings of the two points involved on the event and their neighbours. In order to update the embeddings, a new form of attention, called the Temporal Attention Mechanism, has been developed, inspired by GATs. By considering the recurrence of events that change the embedding of nodes, the

model assumes that it increases the likelihood of a direct connection (Trivedi et al., 2019).

2.3.3 CAW-N

As seen on static graphs, models based on walks have also been designed. Causal Anonymous Walks Network (CAW-N) aims to encode the spatiotemporal structure of dynamic graphs solely by encoding anonymous temporal walks (Wang, Chang, et al., 2021). Temporal walks are random walks that respect the temporal variable in dynamic graphs (Bentert et al., 2020). CAW-N has extended Temporal Walk concept by defining Causal Anonymous Walks (CAW), where after a temporal walk is collected, its nodes are anonymised. The goal is to capture common temporal structures that appear recurrently over time (temporal network motifs) (figure 10)(Wang, Chang, et al., 2021). CAW-N predicts the likelihood of a connection between node u and node v at a timestamp t by performing an arbitrary number of CAWs and encoding each walk using any form of RNN. The resultant set of encodings is passed through a Scaled Dot-Product Attention to forecasting the likelihood of the connection between u and v.

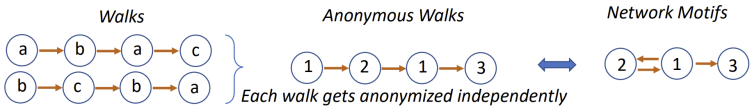


Figure 10: Anonymous Walks proposed by Bentert et al. (2020)

2.3.4 NAT

Luo and P. Li (2022) have proposed a new form to capture the spatiotemporal relation on dynamic graphs, the Neighborhood-Aware Temporal network model (NAT). The novelty relies on the embedding definition, a dictionary-type representation which stores not only the representation of the node but also the representation of its neighbourhood instead of the vector of numbers. Despite the uncommon structure, the GPU manages the dictionary representation thanks to a new data structure that has been created by Luo and P. Li (2022) called N-cache, making it possible for the GPU instructions to manage the complexity of handling hashes. The dictionary stores the current state for each node in different k-hops distances. Once a new edge arrives, all the representations of the edge's node are updated using an RNN module. Finally, when an edge between node u and node v at a timestamp t must be predicted, the aggregation of all representations of the common neighbours of u and v are collected and aggregated (the authors suggest attention to make this operation). The final representation is passed through a neural network responsible for predicting the likelihood.

2.3.5 DyGFormer

Recently, L. Yu et al. (2023) have introduced a new model based on Transformer, DyGFormer. Despite being designed mainly based on Scaled Dot-Product Attention, DyGFormer's novelty lies in how the sequences to represent each node on an edge prediction are defined, having a considerable portion based on a concept of co-occurrence of neighbours.

In order to predict an edge, DyGFormer relies solely on the 1-hop sequence of events involving both nodes of interest (L. Yu et al., 2023). Therefore, each item of the sequence will have a unique representation. The representation of the items in the sequence is a composition of four embeddings:

1. Node embedding: composed of the features of the nodes.
2. Edge embedding: composed of the features of the edge.
3. Time embedding: a positional encoding based on the difference between the event and the time used on the prediction.
4. Co-occurrence embedding: A representation that encodes the frequency of neighbours as their importance. Additionally, the correlation between both nodes might be inferred by comparing the frequencies of similar neighbours between both sequences.

The fourth embedding is a key part of the novelty of DyGFormer in the field. In order to create such embedding, the sequences for each input node u and v must be defined in a particular way:

$$\mathcal{S}_u^t = \{(u,u',t') \mid t' < t)\} \cup \{(u',u,t') \mid t' < t)\}, \tag{6}$$
$$\mathcal{S}_v^t = \{(v,v',t') \mid t' < t)\} \cup \{(v',v,t') \mid t' < t)\}, \tag{7}$$

being u' a neighbour from node u and u' a neighbour from node u.

The co-occurrence of nodes might be calculated as an embedding via a two-layer dense neural network. The fourth embedding for each sequence's item is concatenated, and the sequence of representations passes through a Scaled Dot-Product Attention and then to a classifier to predict the likelihood of the edge. Following this strategy, DyGFormer has achieved remarkable results in the Temporal Graph Benchmark (L. Yu et al., 2023), consistently appearing among the best algorithms on 3 out of 5 datasets currently being the SOTA model (*Leaderboard TGB*, 2024).

2.3.6 GraphMixer

In contrast to the previous methods cited, GraphMixer appears. GraphMixer has presented a strong result on TGB (*Leaderboard TGB*, 2024) motivated by the research question of how good deep learning models might be without using any

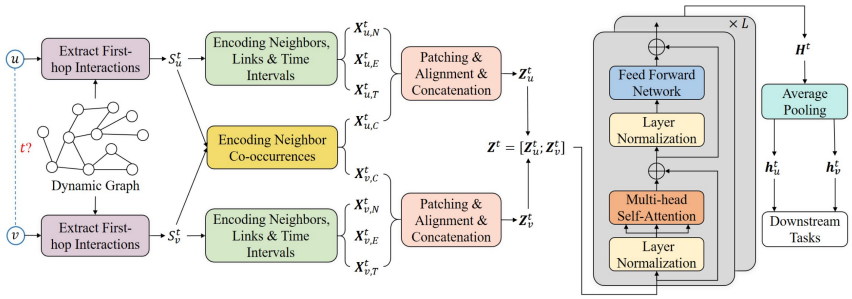


Figure 11: DyGFormer architecture (L. Yu et al., 2023).

form of attention or RNN structure (Cong et al., 2023). Although it only places first in one dataset from the benchmark, the margin by which it outperforms other models is significant.

Graphmixer does not implement any type of RNN or Attention mechanism, though this does not mean it does not rely on any sequence. Graphmixer has been inspired by a previous method from the Computer Vision field called MLP Mixer (Cong et al., 2023). MLP Mixer aims to find an alternative to the frequent usage of CNNs and transformer-based models in the vision field (Tolstikhin et al., 2021). MLP Mixer has defined the Mixer architecture (Figure 12), which, in summary, divides an image into several patches (a hyperparameter of the model). Each patch passes through a dense layer, then the result is transposed, and each row passes through another dense layer. The goal of the first layer is to mix information within the patches, whereas the second mixes information across the patches (Tolstikhin et al., 2021).

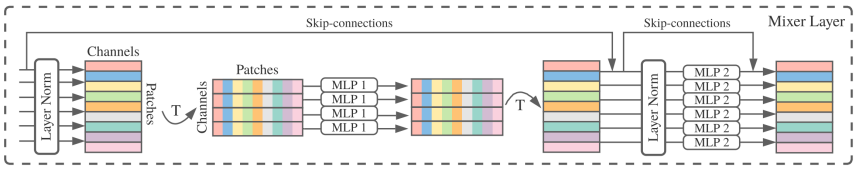


Figure 12: A Mixer Layer - MLP Mixer (Tolstikhin et al., 2021)

Leveraging the concept of MLP Mixer to graphs is the strength of GraphMixer. To predict the likelihood of an edge on a given timestamp t between node u and node v, GraphMixer uses the 1-hop sequence of the events that involved u or v until the timestamp t. A positional encoding is calculated to create an embedding of the time of each item, using a non-trainable function dependent solely on two hyperparameters α and β. The positional embedding is concatenated with the result of one MLP Mixer layer used on the sequence. The result is passed through a classifier, and the likelihood of the edge is predicted (Cong et al., 2023).

2.4 Evaluation of Temporal Networks

In the literature, a consensus has yet to be reached among the papers on which standard metric should be used to evaluate Temporal Network models. Most of the works lean towards reporting Average Precision (AP) (L. Yu et al., 2023)(Cong et al., 2023)(Luo and P. Li, 2022)(Wang, Chang, et al., 2021)(Xu et al., 2020)(Rossi et al., 2020), although some report the area under the curve of the receiver operating characteristic curve (ROC AUC) (Sankar et al., 2020)(Wang, Chang, et al., 2021), Mean Average Rank (Trivedi et al., 2019), or even Accuracy (Xu et al., 2020).

The number of datasets the aforementioned papers have used surpasses 15, making comparisons among different approaches challenging. Thus, aiming to avoid rework and increase comparability from new models, S. Huang et al. (2023) have defined a new benchmark called Temporal Graph Benchmark (TGB). TGB restricts the number of datasets for link prediction tasks to five, all of which are real-world data. Additionally, those datasets have been clustered on the amount of computational resources they demand, divided into three scales as shown in table a.

Scale	Name	#Nodes	#Edges
small	tgbl-wiki-v2	9,227	157,474
small	tgbl-review-v2	352,637	4,873,540
medium	tgbl-coin	638,486	22,809,486
large	tgbl-comment	994,790	44,314,507
large	tgbl-flight	18,143	67,169,570

Table a: Temporal Graph Benchmark Datasets

S. Huang et al. (2023) have also defined a standard metric to evaluate the performance of the model, Mean Reciprocal Rank (MRR). This metric treats the edge link prediction as a rank problem. Moreover, it compares the expected correct edge to be created between node u and node v with other possible edges that should not be created, the negative edges. To calculate MRR, one must measure the likelihood predicted for the positive edge in comparison with the likelihood of the negative edges by sorting the list of likelihoods. The reciprocal rank of the positive edge is the MRR (Croft et al., 2015).

Unfortunately, predicting all the likelihood of the negative edges might be a challenge. For instance, considering the Review dataset (table a), one positive edge connecting two nodes might have 352,636 negative edges. Therefore, TGB has defined an arbitrary list of negative samples created for each row of the datasets for the test and validation split. S. Huang et al. (2023) have carefully selected these lists based on the nodes' history, following the approach of Poursafaei et al. (2022), making the datasets more challenging.

3 Methodology

The following chapter presents the methodology chosen to address the question:

How do specific architectural modifications in dynamic graph models influence the effectiveness of self-attention mechanisms in link prediction tasks?

It will cover the research approach, the research design and the architecture design.

3.1 Research Approach

This dissertation adopts quantitative research focused on implementing architectural changes assessed against datasets from the Temporal Graph Benchmark. In the context of building new machine learning models, a quantitative approach will allow the collection of numerical output from the model, leading to the objective measurement and systematical comparison of multiple model architectures using the performance metrics.

The research is based on the application of existing concepts and established practices in static graphs that can be repurposed to solve the link prediction task in dynamic graphs. By leveraging these techniques, this work builds on the foundation of prior research with a deductive strategy. In addition, the research involves several iterations in the architecture development, model training and validation, with a strong focus on empirical evaluation.

3.2 Research Design

This work is designed with a process that consists of five main stages: dataset selection, data analysis, data pre-processing, architecture design, and architecture validation and evaluation. Here is an overview of these stages:

Dataset Selection: The TGB defines a collection of curated datasets for temporal graph prediction tasks, ensuring standards are met among multiple models. Such datasets have pre-defined splits for training, validation, and testing, guaranteeing consistency with prior research. Two of the datasets from the benchmark have been selected for the dissertation's development.

Data Analysis: The data structure has been evaluated in the data analysis. The timespan by data split was validated, together with the consistency of the data types. Lastly, the distribution of edges and nodes and the distribution of the edges' features have been checked.

Data Pre-processing: Pre-processing has been performed with the focus of preparing the data to allow training, validation, and testing tasks to run with high performance, with the aim of reducing the running time as much as possible. Objects have been created to store the data in efficient ways for the later jobs.

Architecture Design: The architectural changes implemented and tested have used three different strategies and a baseline model. Each approach has been evaluated in isolation and one combination has been explored to determine if integrating strategies improves the results. The models have been iteratively hyperparameter tuned, and each version has been tested using the benchmark datasets.

Architecture Validation and Evaluation: The performance of each model tested has been evaluated using cross-validation, with the pre-defined training and validation split from the benchmark datasets. The evaluation metric is the Mean Reciprocal Rank (MRR), the TGB standard metric.

3.3 Architecture Design

This section presents the details of the four architectures that have been explored to answer how different mechanisms might leverage self-attention to solve link prediction in dynamic graphs. The first one is Baseline architecture, which defines the backbone of the others. The other three extend concepts from static graphs and aim to enhance the Baseline by being plugged into it. Those three strategies are novel approaches in the dynamic graphs field, with Cumulative Message Passing as the most notable.

3.3.1 Baseline

The model is defined into two different sections. The first part is responsible for learning a useful representation, allowing the second part, i.e. the binary classifier, to make good predictions. The base data is a tuple (u, v, t), node u connecting to node v at timestamp t. The edge prediction is a binary classifier: if the tuple represents a real edge, the target y is 1. Otherwise, the target y is 0 if it represents a negative edge. Since the goal is to explore self-attention, the natural choice is to treat the problem as a sequence model. Thus, both node u and node v must be translated to some sequence.

As seen in the literature review, many models have explored different solutions to transform nodes into sequences. Nonetheless, both SOTA models on the TGB leaderboard (*Leaderboard TGB*, 2024), Graphmixer and DyGFormer, use the same sequence definition, events related to nodes from a 1-hop distance. Thus, the Baseline proposed also rely on the same sequence definition (Figure 13). It is worth mentioning that sequences based on 2-hop distances were tested during the project's early stages. Unfortunately, due to an infeasible memory footprint, the idea was discarded.

Each node has a sequence composed of observations defined as a tuple with three values for the Review-v2 and 174 values for the Wiki-v2. The first two values are the same for both datasets. The first is the neighbour node, and the second is the timestamp at which the neighbour was connected to that node. The remaining values (1 for the Review-v2 and 172 for the Wiki-v2) are the features of the edge.

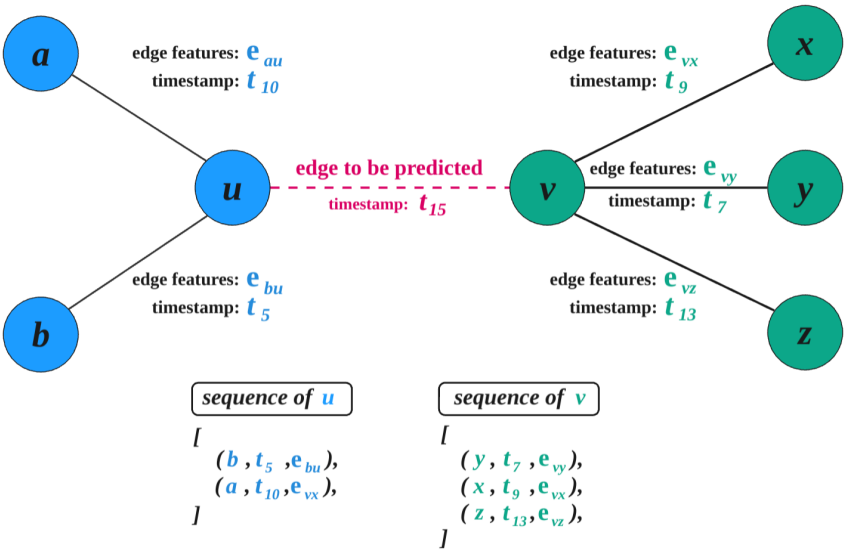


Figure 13: From the dynamic graph to the sequences

The Baseline model relies on three embedding functions applied to each sequence item: node embedding, edge embedding, and time embedding. The node and edge embedding are defined using only one linear dense layer. Time embedding is defined as L. Yu et al. (2023), a linear function followed by a function that alternately applies sines and cosines to the results. The time embedding is then normalised by the square root of the dimension of the embedding's length.

The node embedding is summed with the edge embedding, and the result is concatenated with the time embedding. Then, the two sequences are concatenated to create a more extended sequence that combines both items. A Transformer module (Vaswani et al., 2017) is applied twice, with the first output being the second's input. The stack of the two modules returns a representation for each item on the sequence. This representation is averaged to get a final embedding of the sequence. The model described above is the Encoder of the architecture.

The Encoder is followed by the Decoder, which is defined as two dense layers connected by a non-linear function (RELU). The result of the second layer is passed through a sigmoid since the goal is to create a binary classifier. The complete diagram of the baseline architecture is in Figure 14.

3.3.2 Temporal Walk Embedding

A summary of this strategy is defining the embedding matrix of the node based on a snapshot of the temporal graph at the latest t on the train set. It differs from discrete strategies presented in the literature review because it combines this static version with the concept of Temporal Walks.

The combination of the work from Perozzi et al. (2014) and Wang, Chang,

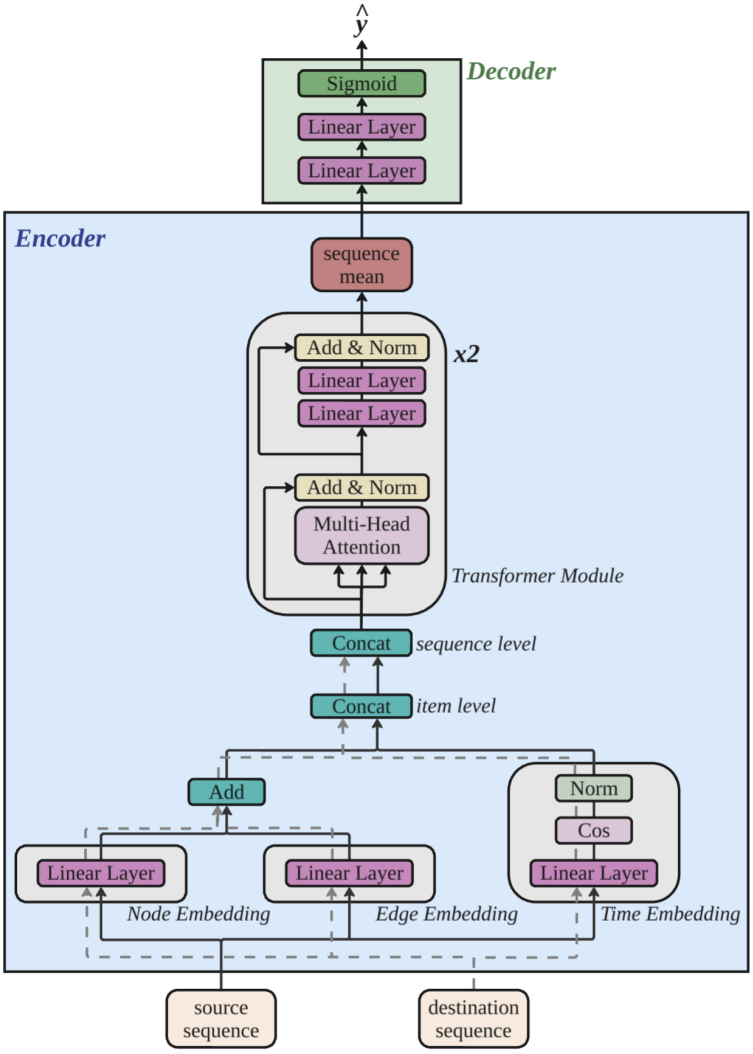


Figure 14: The Baseline model architecture

et al. (2021) is the key to understanding this strategy. As a simplified version of CAW-N, an arbitrary number of random walks will be performed, but only until the 2-hop distance. Adapting the idea from DeepWalk to a dynamic graph, a new dataset will be created using these walks' source and final nodes. The network will receive the source and try to predict the final node. By doing so, the network is an approximation of the conditional probability of seeing the final node on a temporal walk starting on the source node (Perozzi et al., 2014). A simple network using two layers of Graph Attention Network have been used to calculate the node embeddings.

The loss function of the model will be the log softmax of the dot product between the embedding of the source and the embedding of the final node. The rationale behind this is that the network will align both embeddings as closely as possible. For each epoch, the examples that occur more frequently, i.e. those encountered more often during random walks, will be pushed closer together compared to the rarer events.

Two details must be addressed. Firstly, this calculation might be infeasible if all the negative nodes are considered on the denominator of the log softmax. Therefore, a sample of the negative nodes should be taken, with at least the same size as the ones defined by the TGB. By guaranteeing that the sample size is big enough, and the number of random walks is also reasonable, the approximation using the sample of the negative nodes should be sufficiently (Poursafaei et al., 2022). Secondly, to avoid overfitting, after each epoch, the log softmax of the dot product between nodes must be computed using a validation dataset. The model with the lowest validation loss is used to create the final node embeddings. This validation dataset will be generated similarly to the training dataset but will be based on the temporal walks made on a subgraph containing only the validation edges.

3.3.3 Cumulative Message Passing - CMP

The Cumulative Message Passing (CMP) strategy borrows the Message Passing concept from static graphs to temporal graphs. Each sequence item might be understood as a subgraph containing that edge and all the edges before, allowing a message to be computed in this subgraph. Inspired by the approach defined in Kipf and Welling (2017), the embedding of the central node is the average of all the embeddings of its neighbours on the subgraph together with its own. Per item on the sequence, a different message will be calculated representing the embedding of the central node at that moment (Figure 15).

The Cumulative Message Passing defines a fourth embedding to the baseline architecture. This new embedding is summed together with the edge and node embedding. The rationale for this sum is that all three embeddings represent graph-related concepts. Nonetheless, CMP uniquely incorporates both time and space when defining its embedding. By doing so, it bridges two concepts that have been challenging to connect in previous research: the sequence structure from the or-

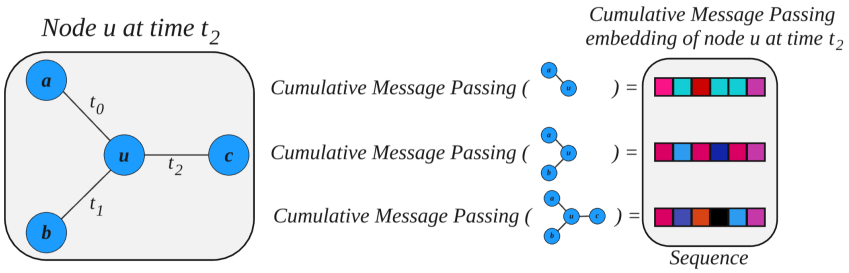


Figure 15: Cumulative Message Passing calculating embedding for each timestamp of a node

ganisation of items and the spatial relationships between items at each point in time

3.3.4 Dual Embedding - DE

The Dual Embedding (DE) strategy is based on the concept discussed in the literature review: directed graphs might perform better with different representations based on their role on the edge, either source or destination (Ou et al., 2016). Therefore, this concept relies on keeping two different embeddings and two different representations for all the nodes and using them accordingly.

A logic to manage dual embedding references must be added. The destination embedding is used for the source node sequence. Following the opposite logic, the source embedding is applied to the destination node sequence.

When combined with the Cumulative Message Passing strategy, the above rule is extended to the fourth embedding. The central node uses its own embedding, with the source embedding applied when the central is the source of the edge and the destination embedding used when it is the edge's destination.

3.3.5 Explored Architectures

The strategies proposed might work in isolation or combined since they are orthogonal to each other. Unfortunately, the cost to test all of the possible combinations is considerably high, and the tight schedule available for the dissertation makes such a task infeasible. Therefore, 5 out of 8 combinations will be kept:

1. *Baseline*
2. *Baseline + Temporal Walk Embedding*
3. *Baseline + Cumulative Message Passing*
4. *Baseline + Dual Embedding*
5. *Baseline + Cumulative Message Passing + Dual Embedding*

Due to restricted time, some combinations had to be excluded. The combinations excluded, and the reasons were the following:

6. *Baseline + Temporal Walk Embedding + Dual Embedding*

Temporal Walk Embedding assumes equality between the nodes since the walk will be made regardless of the node type. Although it is theoretically possible to combine Temporal Walk Embedding and Dual Embedding, further analysis must be conducted to understand precisely how. Therefore, this option has been excluded.

7. *Baseline + Cumulative Message Passing + Temporal Walk Embedding + Dual Embedding*

For the same reason shown in the item *6*, this option was also excluded.

8. *Baseline + Cumulative Message Passing + Temporal Walk Embedding*

Conceptually, this approach might seem reasonable. However, both strategies were inspired by existing concepts from static graphs, and given that there is limited research in the context of static graphs on how message passing could enhance random walks or vice versa, this option was also discarded.

4 Experiments

This section outlines the main tools used in the dissertation, along with a description of the selected dataset. Additionally, it covers the exploratory data analysis, data preprocessing, and the evaluation methods employed.

4.1 Tools and Technologies

The instruments this work relies on include libraries for data collection, processing and visualisation, libraries for graphs specific implementation, and infrastructure that allows heavy data processing.

Python (version 3.11.9) and PyTorch (version 2.4) are the primarily used coding technologies. Python is chosen because it is robust, offers extensive support for scientific computing libraries, and can easily handle complex data structures. Using Python scripts also allowed the models to be run on machines in the cloud. PyTorch is used due to two main reasons. First, it allows for the proficient creation of neural network custom layers; second, for data collection, the TGB has an Application Programming Interface (API), which returns objects using PyTorch geometric data, so using PyTorch eases the integration with the benchmark data.

The libraries used include the Networkx, which can handle graph structure and enables the embedding creation process. The data is pre-processed, analysed, and visualised via well-established libraries capable of dealing with large datasets: Numpy, Pandas, Matplotlib, and Seaborn. Additionally, the Jupyter Notebook was used only to visualise the results.

4.2 Infrastructure

Although the datasets are considered small by TGB, the data volume demands a solid infrastructure to run the training and validation with an acceptable running time. Therefore, in addition to a personal computer equipped with an NVIDIA RTX 4090 GPU, other ad hoc machines were used on the Google Cloud Platform (GCP) with NVIDIA L4 and NVIDIA Tesla P100 GPUs. The Compute Engine (GCE) service for infrastructure from GCP was used via terminal to create and access the machines.

4.3 Datasets selection

The TGB contains a group of five datasets for dynamic graph link prediction, which are divided into small, medium and large. The small datasets are considerably large, a common characteristic of graphs. Given the infrastructure limitations, only the small datasets from the benchmark have been selected for this dissertation: the review-v2 and the wiki-v2.

The review-v2 dataset is a bipartite graph from Amazon reviews containing two types of nodes: people and products. The edge contains information on the

review, with a rating of 1 to 5. The dataset is classified as small by the benchmark definitions and contains 352,637 nodes with 4,873,540 million links. A review from a person on a product is unique and happens at a given timestamp (S. Huang et al., 2023).

The wiki-v2 dataset contains one month of Wikipedia page editions, presenting two types of nodes, authors and pages. The edges of this bipartite network coConcatntain text information from the page updates. In that way, an observation on the dataset is an edition made by a user on a page at a given timestamp. The wiki-v2 is also considered a small dataset by TGB and presents 9,227 nodes and 157,474 edges (S. Huang et al., 2023).

The TGB splits each dataset sequentially into train, validation, and test to ensure that validation and test will be performed the same way by all the applicants. Therefore, the data used in the dissertation already has a pre-defined split of 70% (train), 15% and 15% (validation and test). It is also important to remember that TGB provides 100 negative samples for each true edge on the validation and testing portion of the dataset.

Regarding the availability of the data, the TGB has two main ways to collect the data: directly via its GitHub repository (*GitHub TGB*, 2024) or via a library called py-tgb, which is registered on the public PyPI server (*library TGB*, 2024) . The primary source used was downloading via the library.

4.4 Dataset analysis

The analysis focused on exploring broader characteristics that might guide the development of a new generalist architecture rather than exploiting narrow specificities to achieve tailor-made models. An overall data analysis is as follows.

Firstly, the columns' types were checked, along with missing values and duplicates. Since this is a curated dataset, no data cleaning should be expected to be made. Nonetheless, a minor issue was found. The TGB's library for the review dataset had an entire missing column regarding the features of the edges. The problem was solved by manually downloading this information from the GitHub repository.

Secondly, the timespan was investigated, and the results are in Table b, matching the dataset specifications.

Dataset	Partition Type	From	To
Review-v2	Train	13-06-1999 01:00:00	02-06-2016 01:00:00
	Validation	03-06-2016 01:00:00	06-03-2017 00:00:00
	Test	07-03-2017 00:00:00	04-10-2018 01:00:00
Wiki-v2	Train	day 0	day 20.70
	Validation	day 21.56	day 24.65
	Test	day 26.67	day 29.76

Table b: Table with Validation and Test collected for each dataset by strategy

Additionally, a timestamp versus message features type analysis was conducted to evaluate type consistency and possible precision issues. This analysis is important to draw a conclusion about the model's sequence: whether or not it is possible to put all the elements of a sequence (node, timestamp and edge features) inside the same tensor, sharing the same type. Storing them in separate tensors would increase the complexity of the solution. Based on the analysis in c, for the Review-v2 dataset, the sequence type must be an integer due to the timestamp range, which is consistent with the edge features' data type. Similarly, for the Wiki-v2 dataset, the situation is manageable; although the edge features require a float data type, which has lower precision than integers, it remains acceptable because the timestamps in Wiki-v2 spam a smaller range.

Dataset	Max Timestamp	Edge datatype	Max Precision
Review-v2	1,538,611,200	UInt32	4,294,967,295
Wiki-v2	2,678,373	Float32	16,777,216

Table c: Tensor datatype Analysis - Edge VS Timestamp without losing precision

Then, the distribution of the node degrees was analysed, and the distribution plot is shown in Figure 16.

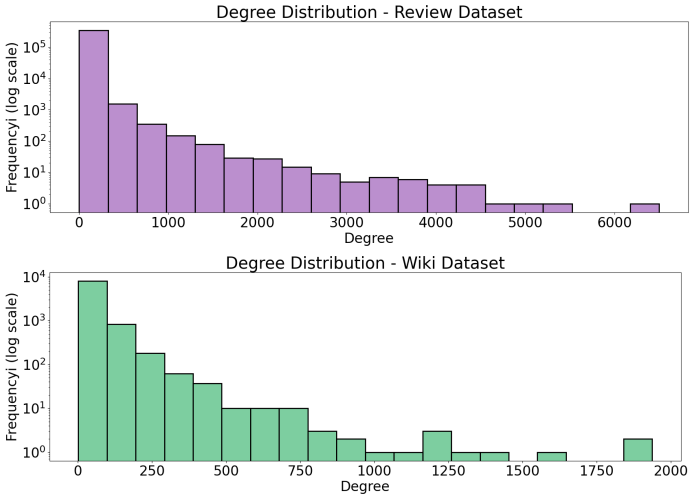


Figure 16: Degree distribution of the Nodes - Wiki-v2 dataset

Since both datasets are bipartite, the analysis of degree was extended to the node types. Figure 17 presents the number of users and the number of products with a specific number of reviews; both are long-tail functions with medians in 11 and 4, and first and third quartiles in 9 and 15 for Users, and 2 and 10 for Products.

The same analysis was made for the wiki dataset (Figure 18), also long-tail functions with medians in 2 and 119, and first and third quartiles in 1 and 8 for Users, and 97 and 166 for Pages.

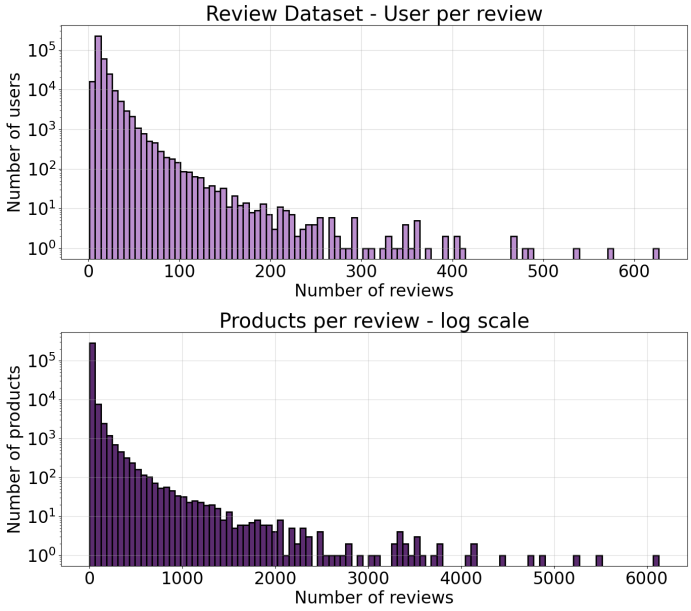


Figure 17: Edits distribution breakdown per User and Page - Wiki-v2 Dataset

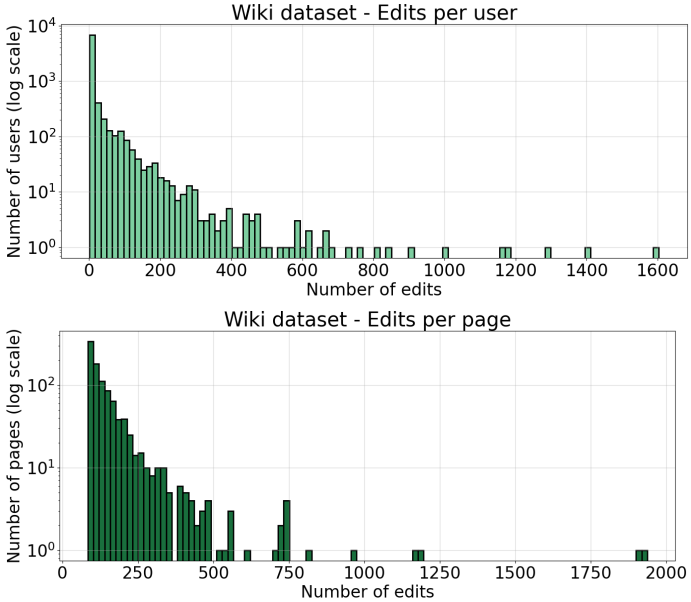


Figure 18: Edits distribution breakdown per User and Page - Wiki-v2 Dataset

The complete analysis can be found in Appendix B.

4.5 Data Pre-processing

The pre-processing phase was critical in ensuring the data was in the appropriate format and structure for the subsequent model training and evaluation tasks. As described below, several key steps have been taken to clean and structure the data effectively.

Data Structure Validation: As the first step of the pre-processing, a validation of the structure of the dataset is performed to ensure all data points are present, correctly filled, and match the expected shape and types. As observed in the review-v2 dataset, the edge messages obtained from the official API were corrupted, resulting in unusable data. A custom function has been implemented to read data directly from a CSV file made available on the TGB website and merge it with the data collected via API, solving the issue in the pre-processing step.

Sequence Creation: The proposed models in this work will depend on sequences of observations. Such sequences need to be created for the actual edges, as well as for the negative samples. During pre-processing, an object is created with all the links a node has in the dataset and their connection timestamp. There will be a sequence for the source of the edge and another for the edge's destination. The sequence is a tridimensional tensor such that the first dimension maps with the dimension of rows on the dataset, the second dimension is the position on the sequence, and the third dimension is for the item's features. On the Pytorch community, this specific tensor is recurrently referred to as *(batch, seq, feature)* (*Pytorch RNN documentation*, 2024). Based on hyperparameter tuning, the sequence has a length 32 for the Review-v2 and 16 for Wiki-v2. Lastly, the features of the sequence items are neighbour node identification, timestamp of the edge and the features of the edge.

Negative Sample Sequences: The sequences used on the negative sample are identical to those used on the positive examples for the equivalent nodes. For instance, a negative example connecting node A with node B will use the last sequence calculated to node A on a previous positive edge and the last sequence calculated to node B on an earlier positive edge. Using a cache for such sequences is crucial to make the dataset manageable in memory. Thus, a cache structure was created, and each negative sample received a key composed of two identifiers: the row ID with the number of the sequence that must be used and if the reference is the source or the destination node.

Preloading Negative Samples: Some implementations treat the negative samples dynamically for validation and testing during model evaluation, which significantly increases computation time. For instance, in the DyGFormer

model, while training on a personal computer with an NVIDIA RTX 4090 took approximately 50 minutes per epoch, validation took up to three hours due to the computational cost of preparing the negative samples in real time. The method used in this work focuses on pre-processing the negative samples before training the model, so all validations reuse the preloaded negative samples. With this strategy, the validation step takes 15 to 25 minutes for the models developed in this work.

Graph Data Preparation: The graph was converted into a Networkx object during pre-processing to train the embeddings. After this step, another function creates the temporal walks using parallelism to run more efficiently. The temporal walks are generated only for training and validation and later used to pre-train embeddings in one of the proposed architectures.

4.6 Architecture Validation and Evaluation

For the evaluation of the models, the validation part of each of the selected datasets was used. The benchmark used in this dissertation provides the dataset with the specific split for validation and testing. For each dataset, 15% of the data is allocated for validation, and 15% is for testing. Additionally, numerous negative samples are generated for each observation, which makes the validation and test datasets considerably large. For instance, the validation portion of the review-v2 contains 100 negative samples per positive one, meaning that around 731,000 actual edges will have over 7 million negative samples. The wiki-v2 validation set presents around 23,600 positive edges and 999 negative samples per observation.

As previously mentioned in the pre-processing subchapter, some steps have been applied to enhance the execution time of the validation, preparing the sequences for both true edges and negative samples. However, with the pre-processing task, running one validation for each epoch would cost around 41 hours of running time for one model. Therefore, a different strategy is used, where, during the model training, a round of validation is executed only after every five epochs. The main drawback of this approach is that after 100 epochs, there are only 20 model evaluations, meaning that if a better model was trained in the middle of a set of five epochs, it is not possible to know that. Still, the strategy allows checking if the model has been overfitted, fulfilling the purpose of cross-validation.

Moreover, an early stopping strategy was applied. If by two consecutive validation runs (10 epochs) the validation does not increase more than 0.001, the early stopping is triggered, resulting in the training stop. The parameters that achieved the highest performance on the validation set is used. Finally, the hyperparameters have been tuned using the validation set of the datasets. They have been tuned in isolation, which means the best set has been chosen based on the baseline architecture. The report of the hyperparameter tuning is the mean and standard deviation of the three highest validation MRR calculated for each hyperparameter.

The measurement uses MRR as a metric. The final MRR of each validation is

an average of all observations predicted. Based on the MRR of each validation, the best model has been selected to finally be evaluated with the test set, which will produce the final performance metric for each of the experimented architectures.

5 Results

Table d Summarizes all the results collected from each strategy; the breakdown of how each number was collected is in the following sub-chapters.

Dataset	Strategy	Validation MRR	Test MRR
Review-v2	Baseline	0.2694	0.2463
	Temporal Walk Embedding	0.2234	0.2102
	Cumulative Message Passing	0.2486	0.2148
	Dual Embedding	0.2734	0.2429
	Cumulative Message Passing + Dual Embedding	0.2847	0.2522
Wiki-v2	Baseline	0.4513	0.3661
	Temporal Walk Embedding	0.4663	0.4141
	Cumulative Message Passing	0.6159	0.5657
	Dual Embedding	0.3874	0.3079
	Cumulative Message Passing + Dual Embedding	0.6171	0.5712

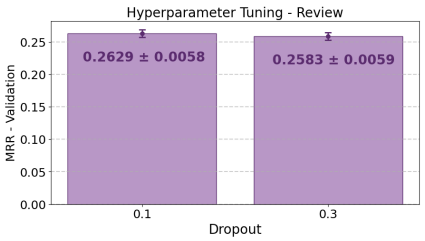
Table d: Table with Validation and Test collected for each dataset by strategy

5.1 Baseline Model

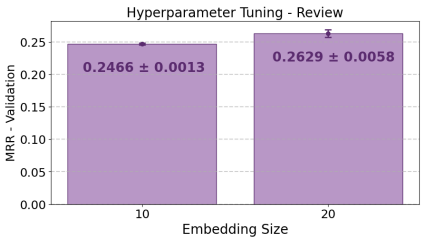
The baseline model's hyperparameters were tuned using both the Review-v2 and Wiki-v2 datasets. The learning rate and batch size were specifically adjusted to fit within the available GPU memory (both cloud and local environments). Other hyperparameters were explored individually, with the remaining held constant.

5.1.1 Review-v2

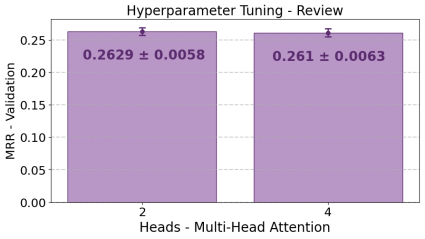
The Hyperparameter tuning for the Review-v2 dataset is presented on the figure 19.



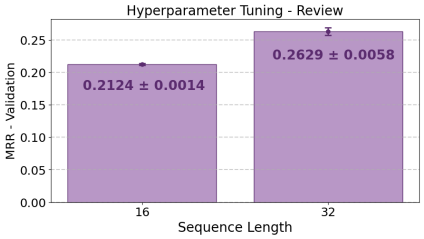
(a) Dropout - Best value: 0.1



(b) Embedding Size - Best value: 20



(c) Number of Heads - Best value: 2



(d) Sequence Length - Best value: 32

Figure 19: Hyperparameter Tuning Baseline Model - Review-v2 dataset

The best values obtained for each hyperparameter are Dropout as 0.1, Embedding Size as 20, 2 as the Number of Heads on Multi-Head Attention, and 32 observations on each sequence. The baseline model set with the aforementioned hyperparameters, together with a learning rate of 0.001 and batch size of 200, was trained, and the loss function with the validation MRR might be seen in the figure 20. The total a

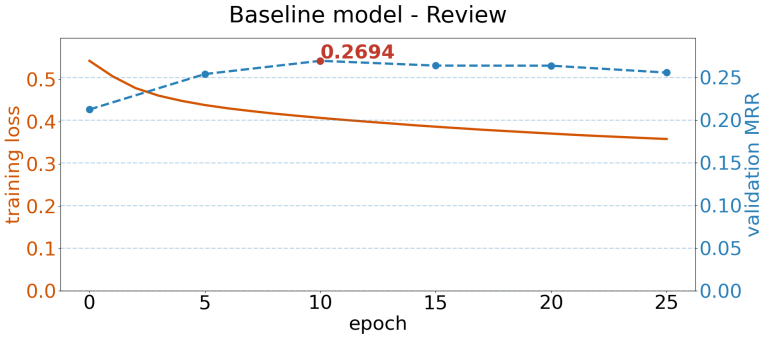
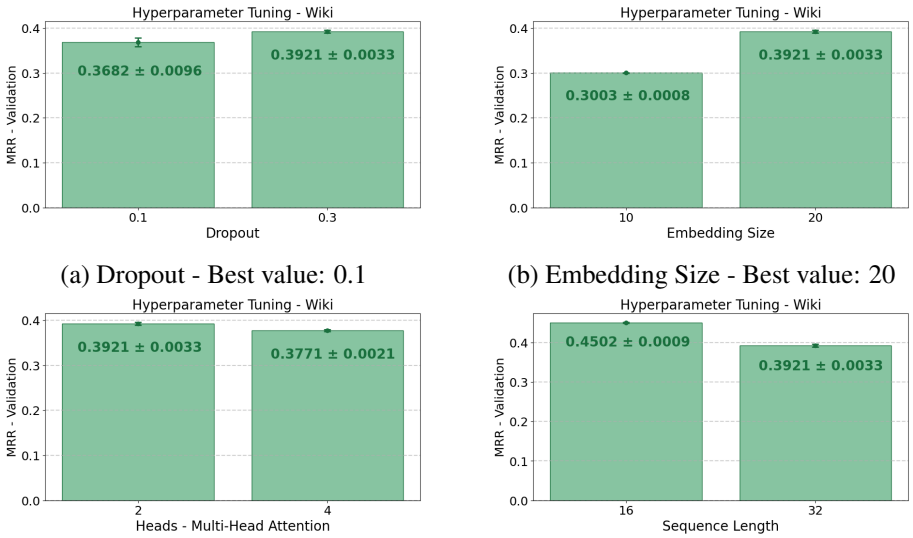


Figure 20: Baseline Model - Review-v2 Dataset

The parameters that achieved the validation MRR of 0.2694 are considered the best model of this section; their training log can be found in Appendix C. The entire training process reported has taken 1 hour and 1 minute to complete. The model achieved 0.2463 of test MRR.

5.1.2 Wiki-v2

The Hyperparameter tuning for the Wiki-v2 dataset is presented on the figure 21.



(a) Dropout - Best value: 0.1

(b) Embedding Size - Best value: 20

(c) Number of Heads - Best value: 2

(d) Sequence Length - Best value: 16

Figure 21: Hyperparameter Tuning Baseline Model - Wiki-v2 dataset

The best values obtained for each hyperparameter are Dropout as 0.1, Embedding Size as 20, 2 as the Number of Heads on Multi-Head Attention, and 16 observations on each sequence. The baseline model set with the aforementioned hyperparameters, together with a learning rate of 0.001 and batch size of 200, was trained, and the loss function with the validation MRR might be seen in the figure 22.

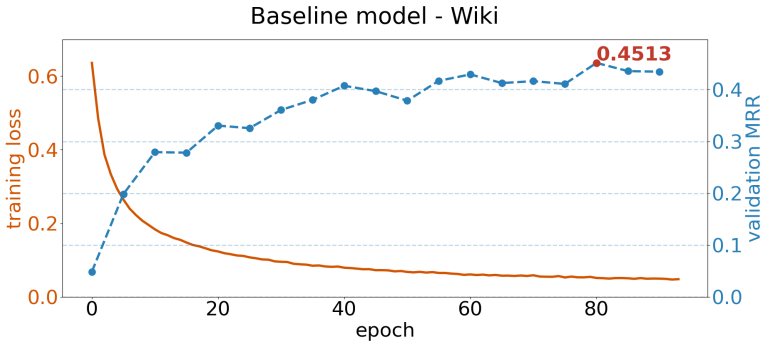


Figure 22: Baseline Model - Wiki-v2 Dataset

The parameters that achieved the validation MRR of 0.4513 are considered the best model of this section; their training log can be found in Appendix I. The entire

training process reported has taken 42 minutes to complete. The model achieved 0.3661 of test MRR.

5.2 Temporal Walk Embedding

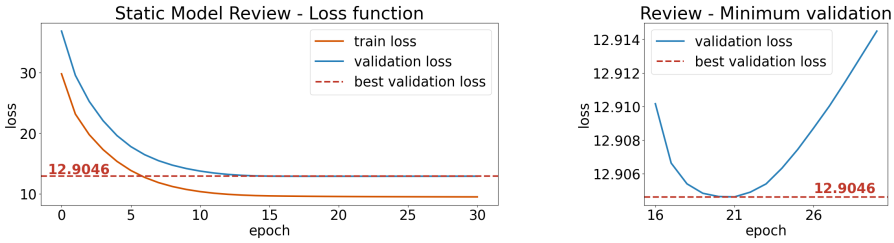
The Temporal Walk Embedding strategy begins with training the static model based on the temporal graph. The static model is trained using temporal walks based on the static graph defined by the training dataset. In contrast, the validation loss is calculated using the temporal walks made on the subgraph defined by the validation dataset.

The model with the lowest validation loss is used to create a static embedding representation of each node, and this representation is passed to the baseline and set as frozen.

First, the static model's training loss and validation metric are presented. Then, the result of the baseline model enhanced with the embedding generated by the static model is presented.

5.2.1 Review-v2

The static model loss trained over the Temporal walks on the training dataset is presented in Figure 23. The log is in Appendix G.



(a) Loss function

(b) Minimum validation zoom

Figure 23: Static model Review-v2 dataset - Model trained with Temporal Walks

The baseline model using the static node representation created from the static model has presented the results in Figure 24.

The parameters that achieved the validation MRR of 0.2234 are considered the best model of this section; their training log can be found in Appendix F. The entire training process reported has taken 5 hours and 27 minutes to complete (4 hours and 17 minutes to the Static Model and 1h10 minutes to the Sequence model). The model achieved 0.2102 of test MRR.

5.2.2 Wiki-v2

The static model loss trained over the Temporal walks on the training dataset is presented in Figure 25. The log is in Appendix M.

This i the model has double the parameters related to the node embedding when compared to the baseline.

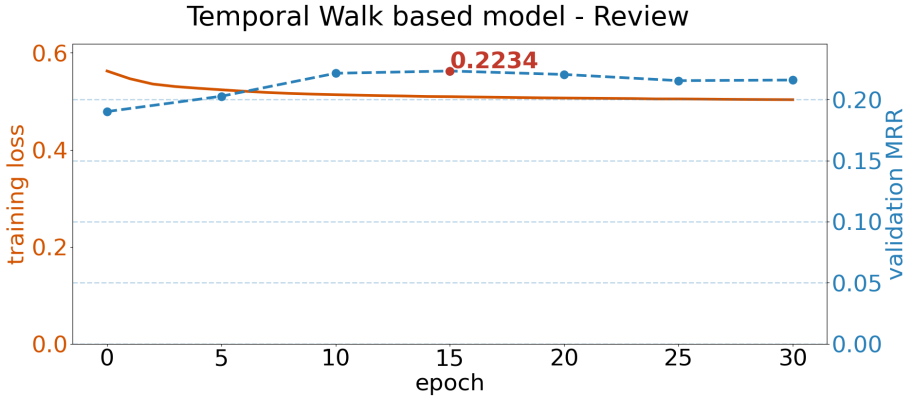
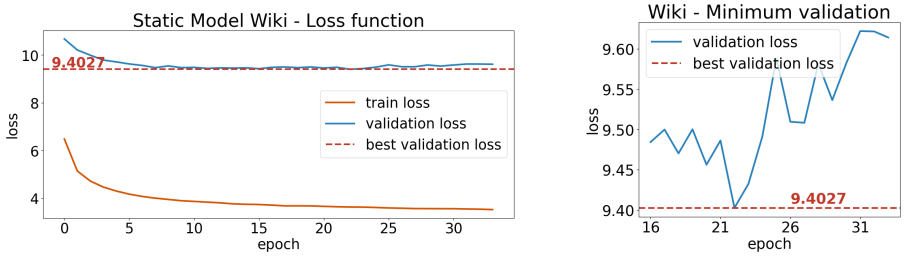


Figure 24: Temporal Walk based model - Review-v2



(a) Loss function

(b) Minimum validation zoom

Figure 25: Static model Wiki-v2 dataset - Model trained with Temporal Walks

The baseline model using the static node representation created from the static model has presented the results in Figure 26.

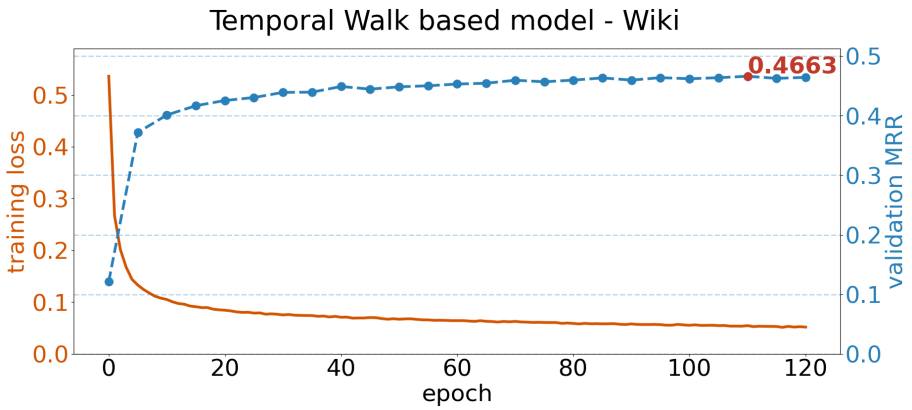


Figure 26: Temporal Walk based model - Wiki-v2

The parameters that achieved the validation MRR of 0.4663 are considered the best model of this section; their training log can be found in Appendix L. The entire training process reported has taken 1 hour and 57 minutes to complete (1h07 minutes to the Static Model and 50 minutes to the Sequence Model). The model achieved 0.4141 of test MRR.

5.3 Cumulative Message Passing

Cumulative Message Passing strategy enhances the baseline with a new embedding inspired on the concept of message passing.

5.3.1 Review-v2

The loss function and the validation MRR seen during the train using the Review-v2 dataset are presented in Figure 27.

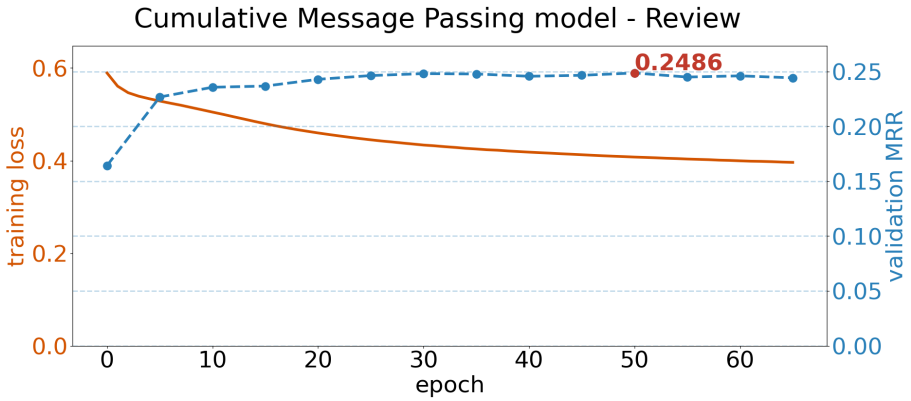


Figure 27: Cumulative Message Passing model - Review-v2

The parameters that achieved the validation MRR of 0.2486 are considered the best model of this section; their training log can be found in Appendix E. The entire training process reported has taken 2 hours and 35 minutes to complete. The model achieved 0.2148 of test MRR.

5.3.2 Wiki-v2

The loss function and the validation MRR seen during the train using the Wiki-v2 dataset are presented in Figure 28.

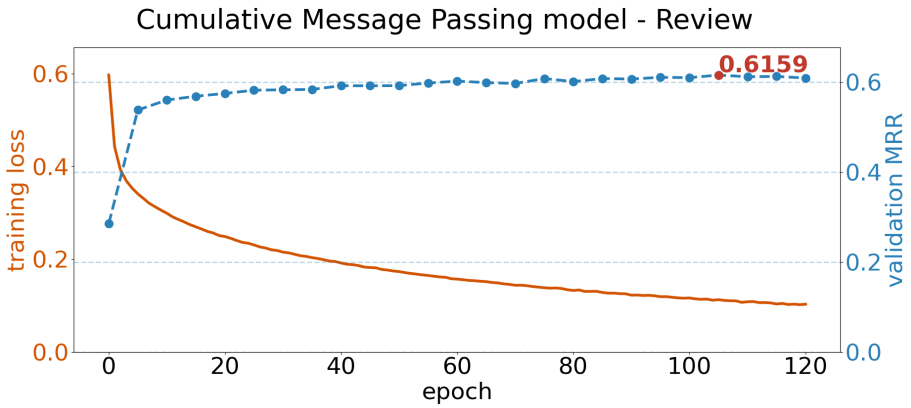


Figure 28: Cumulative Message Passing model - Wiki

The parameters that achieved the validation MRR of 0.6159 are considered the best model of this section; their training log can be found in Appendix K. The entire training process reported has taken 2 hours and 8 minutes to complete. The model achieved 0.5657 of test MRR.

5.4 Dual Embedding

The Dual Embedding model is the baseline model with two embeddings for the nodes, one for the source nodes and another for the destination nodes.

5.4.1 Review-v2

The loss function and the validation MRR seen during the train using the Review dataset are presented in Figure 29.

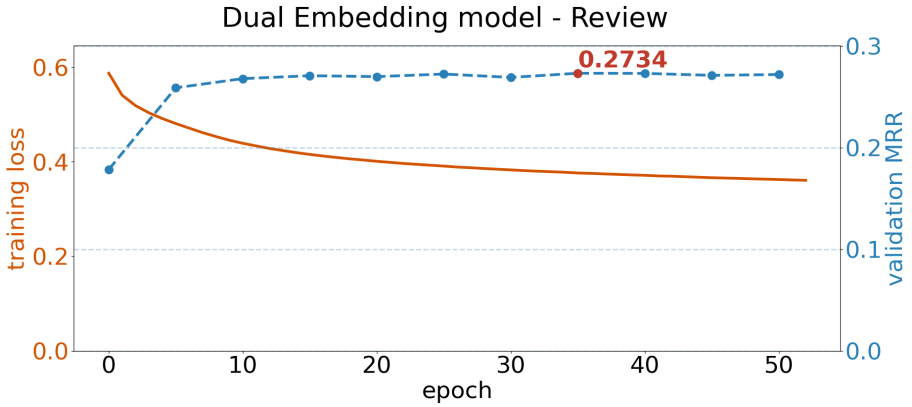


Figure 29: Dual Embedding Model - Review-v2

The parameters that achieved the validation MRR of 0.2734 are considered the best model of this section; their training log can be found in Appendix D. The entire training process reported has taken 1 hour and 59 minutes to complete. The model achieved 0.2429 of test MRR.

5.4.2 Wiki-v2

The loss function and the validation MRR seen during the train using the Wiki-v2 dataset are presented in Figure 30.

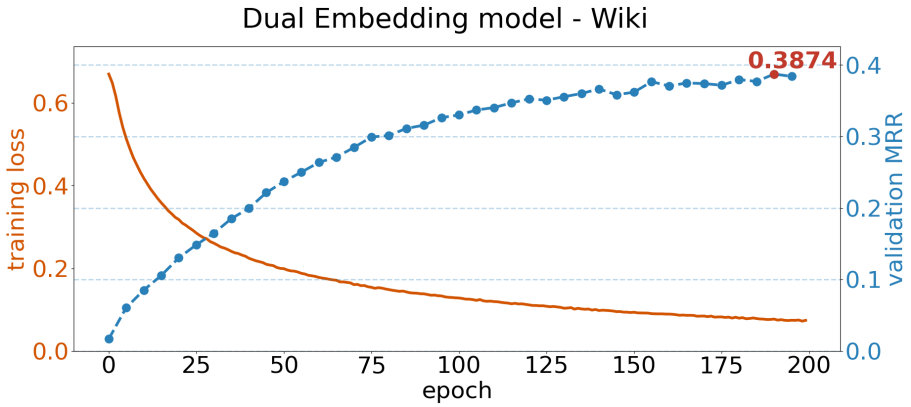


Figure 30: Dual Embedding Model - Wiki-v2

The parameters that achieved the validation MRR of 0.3874 are considered the best model of this section; their training log can be found in Appendix J. The entire training process reported has taken 3 hours and 36 minutes to complete. The model achieved 0.3079 of test MRR.

5.5 Cumulative Message Passing + Dual Embedding

The Cumulative Message Passing + Dual Embedding strategy combines both strategies to define the architectu

5.5.1 Review-v2

The loss function and the validation MRR seen during the train using the Review-v2 dataset are presented in Figure 31.

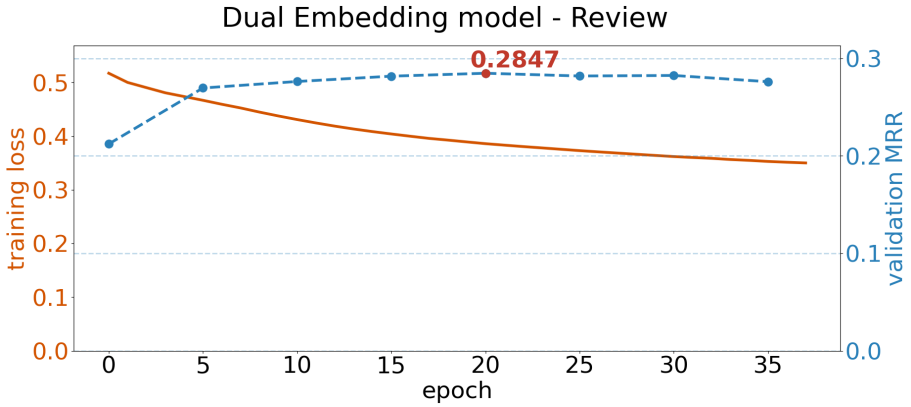


Figure 31: Cumulative Message Passing + Dual Embedding model - Review-v2

The parameters that achieved the validation MRR of 0.2847 are considered the best model of this section; their training log can be found in Appendix H. The

entire training process reported has taken 2 hours and 51 minutes to complete. The model achieved 0.2522 of test MRR.

5.5.2 Wiki-v2

The loss function and the validation MRR seen during the train using the Wiki-v2 dataset are presented in Figure 32.

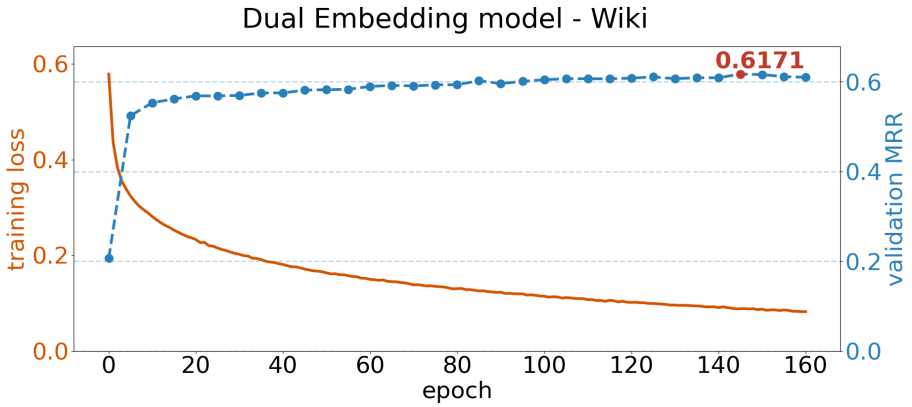


Figure 32: Cumulative Message Passing + Dual Embedding model - Wiki-v2

The parameters that achieved the validation MRR of 0.6171 are considered the best model of this section; their training log can be found in Appendix H. The entire training process reported has taken 4 hours and 19 minutes to complete. The model achieved 0.5712 of test MRR.

5.6 Comparison to Benchmark

Table e and Table f present all the strategies proposed here together with the leaderboard of TGB *Leaderboard TGB*, (2024) to understand where they are situated among the other models. Test MRR is the metric used to rank.

Table e: Review-v2 - TGB Leaderboard (*Leaderboard TGB*, 2024), together with the methods proposed

TGB rank	Method	Test MRR	Validation MRR
1	GraphMixer	0.521 ± 0.015	0.428 ± 0.019
2	TGAT	0.355 ± 0.012	0.324 ± 0.006
3	TGN	0.349 ± 0.020	0.313 ± 0.012
4	NAT	0.341 ± 0.020	0.302 ± 0.011
	CMP + DE	**0.2522**	**0.2847**
	Baseline	**0.2463**	**0.2694**
	DE	**0.2429**	**0.2734**
5	DyGFormer	0.224 ± 0.015	0.219 ± 0.017
6	DyRep	0.220 ± 0.030	0.216 ± 0.031
	CMP	**0.2148**	**0.2486**
	Temporal Walk Embedding	**0.2102**	**0.2234**
7	CAWN	0.193 ± 0.001	0.200 ± 0.001
8	TCL	0.193 ± 0.009	0.199 ± 0.007
9	EdgeBank(tw)	0.025	0.024
10	EdgeBank(unlimited)	0.023	0.023

Table f: Wiki-v2 - TGB Leaderboard (*Leaderboard TGB*, 2024), together with the methods proposed

TGB rank	Method	Test MRR	Validation MRR
1	DyGFormer	0.798 ± 0.004	0.816 ± 0.005
2	NAT	0.749 ± 0.010	0.773 ± 0.011
3	CAWN	0.711 ± 0.006	0.743 ± 0.004
	CMP + DE	**0.5712**	**0.6171**
4	EdgeBank(tw)	0.571	0.600
	CMP	**0.5657**	**0.6159**
5	TGR-TGN	0.531 ± 0.011	0.577 ± 0.012
6	EdgeBank(unlimited)	0.495	0.527
	Temporal Walk Embedding	**0.4141**	**0.4663**
7	TGN	0.396 ± 0.060	0.435 ± 0.069
	Baseline	**0.3661**	**0.4513**
	DE	**0.3079**	**0.3874**
8	TCL	0.207 ± 0.025	0.198 ± 0.016
9	TGAT	0.141 ± 0.007	0.131 ± 0.008
10	GraphMixer	0.118 ± 0.002	0.113 ± 0.003
11	DyRep	0.050 ± 0.017	0.072 ± 0.009

6 Discussion

The thesis aims to understand how architecture modifications in dynamic graph models enhance their self-attention mechanism in link prediction tasks. Four model architectures have been designed to explore the research question based on three different strategies. Cumulative Message Passing, one of the proposed models, is a novel approach that achieves solid results.

Several aspects of self-attention are assessed and discussed in the further subsections. The list of the findings is the following:

1. Using solely the 1-hop distance of a node to define its sequence, together with a simple Transformed approach, achieves solid performance on the benchmark, suppressing several previous studies.

2. Temporal Walks, as used in this work to encode the temporal topology of the nodes, achieved a subpar performance.

3. Strategies that aim to encode the neighbourhood of a node considerably boost self-attention to forecast link prediction.

4. On directed graphs, using two different embeddings for the nodes, one for the source and another for the edge's destination might improve the model's performance.

5. Combining strategies, such as the combination of Dual Embedding and Cumulative Message Passing, may boost performance and, in this work, resulted in the best strategy found.

6.1 Baseline

The baseline model demonstrates a strong performance even compared to well-established models on the leaderboard Table e and Table f, such as GraphMixer or TGAT. It has two main strengths: the Transformer model and the 1-hop sequence.

The Transformer being a robust model is well-discussed in the literature (Wolf et al., 2020)(Dosovitskiy et al., 2021)(L. Yu et al., 2023)(Carion et al., 2020), and this work once more indicates the same. Nonetheless, the Transformer relies on some sequence definition, and Dynamic Graphs can be defined as a sequential problem. It is important to note that the sequence on Dynamic Graphs might be defined in a variety of ways. In this work, the sequence used is based on the 1-hop distance of the nodes. Other research has explored the same approach (Cong et al., 2023)(L. Yu et al., 2023).

Regardless, previous works have not tested the effectiveness of the 1-hop sequence alone. By defining a Baseline, this work highlights that a model can achieve high performance by only using the 1-hop distance and a simple implementation of the Transformer. The Baseline is better than the fifth position of the Benchmark for Review-v2 and the eighth position on Wiki-v2.

Even compared with the other strategies proposed in the dissertation, the Baseline is the second-best strategy on Review-v2 and the fourth-best strategy on Wiki-v2. It is also worth noting that it was the fastest strategy to train.

6.2 Temporal Walk Embedding

The Temporal Walk Embedding has been defined as a strategy to pass the 2-hop distance information of the graph's nodes to the model without relying on a 2-hop sequence. This strategy is used because the computational cost of 2-hop distance is expensive (L. Yu et al., 2023). The combination of the findings from (Perozzi et al., 2014) and (Hamilton et al., 2017) inspired the idea of encoding the conditional probabilities of nodes on a Temporal Walk.

Nevertheless, using Temporal Walks, as proposed in this work, has proven to be the weakest presented strategy. It barely improved the Baseline performance of the benchmark on Wiki-v2 and reduced the performance on Review-v2.

Temporal Walk Embedding introduces a heavy complexity to the model. The strategy assumes handling two different models simultaneously, creating an intermediate dataset with the Temporal Walks information, and performing two distinct model training. Thus, its final performance does not justify its use.

6.3 Cumulative Message Passing

The Cumulative Message Passing (CMP) is a novel idea based on the concept of Message Passing (Gilmer et al., 2017). CMP creates a new embedding founded on the Message Passing framework applied to the subgraphs defined on each timestamp of the sequence.

CMP achieves substantial results on Wiki-v2. On Review-v2, its performance is subpar, although when combined with the Dual Embedding, it gets the best result among the proposed strategies. Therefore, CMP underscores that encoding a node's neighbourhood strengthens the self-attention performance.

It is worth to note that CMP increases the training time since it introduces a new set of parameters to the model. Additionaly, the approach sheds light on a possible and simple way to introduce well-established concepts from static graphs onto dynamic graph models.

6.4 Dual Embedding

The Dual Embedding (DE) strategy is based on managing two different node embeddings on directed graphs, one for the destination nodes and another for the source nodes.

The addition of a new embedding considerably increases the model's training time due to doubling the number of parameters related to node embedding. Its performance in Review-v2 and Wiki-v2 datasets is slightly worse than the Baseline.

Yet, by combining it with CMP, it achieves the best performance on both. Suggesting that, depending on the dataset, the performance of self-attention models might be increased by using two different node embeddings.

The impact of a dual embedding emphasised here is similar to the one seen in the research of (Ou et al., 2016). Nonetheless, this dissertation defines the Dual Embedding concept in the context of dynamic graphs, bringing a different lens to the solution.

6.5 Combined Strategies

Lastly, there is a combination of strategies. This dissertation combines the strategies solely in one way due to time and resource limitations. Nevertheless, having at least one combination of strategies highlights the usefulness of handling such solutions as modules and combining them based on need. Previous research presents their ideas as isolated models and isolated solutions rather than pluggable modules, which, by definition, reduces their future usability.

When strategies can be combined, as in Dual Embedding and Cumulative Message Passing, performance may increase considerably, which is a valid approach. The combination of CMP and DE results in the best performance on both datasets and for the problems posed, it is the preferred strategy defined.

6.6 Limitations

The scope of the experimented architectures and the number of datasets used from the benchmark are the main limitations of this work.

6.6.1 Limitations on the scope

This research focused mainly on node embedding strategies and how they affect the overall performance of a self-attention model. Nonetheless, other scopes should be explored further.

Unexplored scopes of this research were:

- The sequences assigned to the nodes and how different approaches affect the model's performance.
- The impact of different time embeddings, trainable and not-trainable, on the model's accuracy.
- Exploring other self-attention mechanism rather than Scaled-Dot product.
- Investigating other strategies of Multi-Head Attention, tailored for graph problems.
- Identifying strategies that train quickly and efficiently, despite the complexity of graph data, while achieving strong results.

6.6.2 Limitations on the dataset

Only the two datasets considered small by the benchmark were used due to resource limitations. To enhance the accuracy and consistency of the findings presented, further exploration of the proposed strategies must be conducted on the remaining datasets.

The datasets have some core differences, such as one being a multigraph and the other not, the spamming time being quite distinct, and the volume of edges being considerably different. Nevertheless, they share some structural similarities. Both datasets are Directed and Bipartite Graphs, which can introduce a bias on the significance of the findings. Exploring the architectures further on a variety of datasets is a beneficial step towards ensuring the reliability of the findings.

6.7 Advice for future research

One must notice that Graph problems are highly resource-intensive in all aspects: Processing (CPU and GPU), memory (RAM and VRAM) and storage. Unfortunately, having a well-equipped environment is mandatory for exploring solutions and diving into new ideas effectively.

Therefore, expending time in preprocessing results and being mindful of the time and space complexity of all the procedures designed is also required. Poorly designed algorithms running over graphs fastly might become intractable, even while running on the most powerful infrastructures.

7 Conclusion

This thesis has explored how specific architecture modifications in dynamic graph models influence the effectiveness of self-attention mechanisms in link prediction tasks. Several strategies have been explored via a quantitative approach based on an external benchmark, the Temporal Graph Benchmark (TGB) (S. Huang et al., 2023). By using an external benchmark, all the findings can be compared directly to previous methods in the field. Also, the proposed architecture design is defined as a set of unpluggable pieces on top of a self-attention baseline model, enabling comparison of effectiveness by numerical evaluation.

One of the highlights of the work is a novel approach to building node embeddings on dynamic graphs, the Cumulative Message Passing (CMP). CMP leverages the Message Passing concept (Gilmer et al., 2017) from static graphs to dynamic graphs. Its core idea is based on aggregating the temporal information of the nodes across time. Besides achieving solid performance on the link-prediction datasets from TGB, CMP demonstrated that incorporating neighbourhood information on the representation of the node passed to self-attention might boost its performance in dynamic graph models.

The baseline self-attention model in this research is built entirely around the 1-hop distance sequence of nodes involved in edge prediction. This approach outperformed many other models in the benchmark, highlighting the strength of using 1-hop distance interactions to define the sequence. Although several studies have employed this strategy (L. Yu et al., 2023)(Cong et al., 2023), to the best of my knowledge, it has never been tested in isolation as it was in this research. Demonstrating how relevant is the sequence definition on self-attention based models.

The dual embedding strategy was defined based on implementing different embedding for source and destination nodes on directed graphs. This method brought some boost to one of the datasets, suggesting that the work from Ou et al. (2016) might be extended to dynamic graphs.

Lastly, a strategy employing Temporal Walks aiming to bridge previous research on static graphs to dynamic graphs has been designed. Although it presented the weakest performance among the options analysed, similar research using concepts related to Temporal Walks has achieved solid results on the benchmark, suggesting that there is still area for further exploration.

Another important aspect of this work is the modular layout. By defining the modules as pluggable parts and measuring the impact of each one, the significance of each piece of the architecture became clear. In addition, this approach increases collaborative work, as each component proposed here might be used independently in future research.

In terms of limitations, this dissertation accounts for two main ones. The scope of the research was focused primarily on node embedding strategies, leaving other areas unexplored. Additionally, the study was restricted only to two datasets from the Temporal Graph Benchmark due to resource limitations, which impact the generalisation of the findings.

This dissertation contributes to the field of dynamic graph models by proposing and evaluating new architectural designs. While the Baseline model's simplicity provided solid performance, more complex strategies like Cumulative Message Passing and Dual Embedding offered additional improvements. The findings underscore the potential of modular architectures in dynamic graphs and showcase the potential for further exploration into combining strategies to optimise self-attention models.

Plenty of room is left for future research. Firstly, different approaches to sequence definition might help to enhance the performance of the self-attention mechanism, especially approaches that consider farther nodes. Secondly, a broad study of time embedding should be conducted to understand the impact of trainable and non-trainable approaches, specifically on dynamic graphs. Thirdly, exploring different strategies of self-attention is another relevant topic, as they might interact with the embeddings in different ways. Finally, all the findings must be evaluated and expanded to a broader set of datasets.

Dynamic graphs are an exciting field that is melded into multiple solutions people interact with daily, from recommendation systems that impact one's decisions to the navigation system used to commute. Exploring advancements in deep learning together with graph-based solutions can benefit civilisation by making it more connected and aware of the whole. Multiple fields, such as biological systems, climate science, and astronomy, might be modelled and explored using graphs, showcasing the broad impact that focusing on graph advancements can bring to society.

References

Bentert, M., Himmel, A.-S., Nichterlein, A., and Niedermeier, R., 2020. Efficient computation of optimal temporal walks under waiting-time constraints. *Applied network science* [Online], 5(1), p.73. Available from: https://doi.org/10.1007/s41109-020-00311-0.

Bodnar, C., Di Giovanni, F., Chamberlain, B., Lió, P., and Bronstein, M., 2022. Neural sheaf diffusion: a topological perspective on heterophily and oversmoothing in gnns. S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, eds. *Advances in neural information processing systems* [Online]. Vol. 35. Curran Associates, Inc., pp.18527–18541. Available from: https://proceedings.neurips.cc/paper_files/paper/2022/file/75c45fca2aa416ada062b26cc4fb7641-Paper-Conference.pdf.

Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P., 2017. Geometric deep learning: going beyond euclidean data. *Ieee signal processing magazine* [Online], 34(4), pp.18–42. Available from: https://doi.org/10.1109/MSP.2017.2693418.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S., 2020. End-to-end object detection with transformers. A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds. *Computer vision – eccv 2020*. Cham: Springer International Publishing, pp.213–229.

Cho, K., Merriënboer, B. van, Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y., 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. A. Moschitti, B. Pang, and W. Daelemans, eds. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* [Online]. Doha, Qatar: Association for Computational Linguistics, pp.1724–1734. Available from: https://doi.org/10.3115/v1/D14-1179.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *Nips 2014 workshop on deep learning, december 2014*.

Cong, W., Zhang, S., Kang, J., Yuan, B., Wu, H., Zhou, X., Tong, H., and Mahdavi, M., 2023. Do we really need complicated model architectures for temporal networks? *The eleventh international conference on learning representations* [Online]. Available from: https://openreview.net/forum?id=ayPPc0SyLv1.

Croft, W.B., Metzler, D., and Strohman, T., 2015. Evaluating search engines. *Search engines: information retrieval in practice*. Person Education, pp.308–322.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N., 2021. An image is worth 16x16 words: transformers for image recognition at scale. *International conference on learning representations* [Online]. Available from: https://openreview.net/forum?id=YicbFdNTTy.

Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., and Yin, D., 2019. Graph neural networks for social recommendation. *The world wide web conference* [Online], Www '19. San Francisco, CA, USA: Association for Computing Machinery, pp.417–426. Available from: https://doi.org/10.1145/3308558.3313488.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y.N., 2017. Convolutional sequence to sequence learning. D. Precup and Y.W. Teh, eds. *Proceedings of the 34th international conference on machine learning* [Online]. Vol. 70, Proceedings of machine learning research. PMLR, pp.1243–1252. Available from: https://proceedings.mlr.press/v70/gehring17a.html.

Gers, F., Schmidhuber, J., and Cummins, F., 1999. Learning to forget: continual prediction with lstm. *1999 ninth international conference on artificial neural networks icann 99. (conf. publ. no. 470)* [Online]. Vol. 2, 850–855 vol.2. Available from: https://doi.org/10.1049/cp:19991218.

Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., and Dahl, G.E., 2017. *Neural message passing for quantum chemistry* [Online]. arXiv: 1704.01212 [cs.LG]. Available from: https://arxiv.org/abs/1704.01212.

GitHub TGB, 2024. https://github.com/shenyangHuang/TGB/. [Accessed 05-09-2024].

Goebel, K. and Kirk, W.A., 1990. Banach's contraction principle. *Topics in metric fixed point theory*. Cambridge University Press, pp.7–26.

Goodfellow, I., Bengio, Y., and Courville, A., 2017a. Autoencoders. *Deep learning*. The MIT Press, pp.493–517.

Goodfellow, I., Bengio, Y., and Courville, A., 2017b. Representation learning. *Deep learning*. The MIT Press, pp.517–547.

Grover, A. and Leskovec, J., 2016. Node2vec: scalable feature learning for networks. *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* [Online], Kdd '16. San Francisco, California, USA: Association for Computing Machinery, pp.855–864. Available from: https://doi.org/10.1145/2939672.2939754.

Guillemin, V. and Pollack, A., 1974. Manifolds and smooth maps. *Differential topology*. Prentice-Hall, Inc, pp.1–56.

Hamilton, W.L., 2020a. Convolution-inspired gnns. *Graph representation learning*. Morgan and Claypool, pp.85–88.

Hamilton, W.L., 2020b. Neighborhood reconstruction methods. *Graph representation learning*. Morgan and Claypool, pp.29–37.

Hamilton, W.L., 2020c. Spectral graph convolutions. *Graph representation learning*. Morgan and Claypool, pp.81–84.

Hamilton, W.L., Ying, R., and Leskovec, J., 2017. Inductive representation learning on large graphs. *Nips*.

Harris, J., Hirst, J., and Michael, M., 2008. Introductory concepts. *Combinatorics and graph theory*. Springer-Verlag, pp.1, 17.

Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. *Neural comput.* [Online], 9(8), pp.1735–1780. Available from: https://doi.org/10.1162/neco.1997.9.8.1735.

Huang, S., Poursafaei, F., Danovitch, J., Fey, M., Hu, W., Rossi, E., Leskovec, J., Bronstein, M., Rabusseau, G., and Rabbany, R., 2023. Temporal graph benchmark for machine learning on temporal graphs. A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, eds. *Advances in neural information processing systems* [Online]. Vol. 36. Curran Associates, Inc., pp.2056–2073. Available from: https://proceedings.neurips.cc/paper_files/paper/2023/file/066b98e63313162f6562b35962671288-Paper-Datasets_and_Benchmarks.pdf.

Huang, Z., Sun, Y., and Wang, W., 2020. Learning continuous system dynamics from irregularly-sampled partial observations. *Proceedings of the 34th international conference on neural information processing systems*, Nips '20. Vancouver, BC, Canada: Curran Associates Inc.

Kipf, T.N. and Welling, M., 2017. Semi-supervised classification with graph convolutional networks. *International conference on learning representations (iclr)*.

Leaderboard TGB, 2024. https://tgb.complexdatalab.com/docs/leader_linkprop/. [Accessed 20-08-2024].

Li, X., Zhang, M., Wu, S., Liu, Z., Wang, L., and Yu, P.S., 2020. Dynamic graph collaborative filtering. *2020 ieee international conference on data mining (icdm)* [Online]. Los Alamitos, CA, USA: IEEE Computer Society, pp.322–331. Available from: https://doi.org/10.1109/ICDM50108.2020.00041.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R., 2017. *Gated graph sequence neural networks* [Online]. arXiv: 1511.05493 [cs.LG]. Available from: https://arxiv.org/abs/1511.05493.

Library TGB, 2024. https://pypi.org/project/py-tgb/. [Accessed 05-09-2024].

Luo, Y. and Li, P., 2022. Neighborhood-aware scalable temporal network representation learning. *Learning on graphs conference*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., and Dean, J., 2013. Distributed representations of words and phrases and their compositionality. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, eds. *Advances in neural information processing systems* [Online]. Vol. 26. Curran Associates, Inc. Available from: https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

Ou, M., Cui, P., Pei, J., Zhang, Z., and Zhu, W., 2016. Asymmetric transitivity preserving graph embedding [Online], Kdd '16. San Francisco, California, USA: Association for Computing Machinery, pp.1105–1114. Available from: https://doi.org/10.1145/2939672.2939751.

Perozzi, B., Al-Rfou, R., and Skiena, S., 2014. Deepwalk: online learning of social representations. *Proceedings of the 20th acm sigkdd international conference on knowledge discovery and data mining* [Online], Kdd '14. ACM. Available from: https://doi.org/10.1145/2623330.2623732.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P., 2021. Learning mesh-based simulation with graph networks. *International conference on learning representations (iclr)* [Online]. Available from: https://openreview.net/forum?id=roNqYL0_XP.

Poursafaei, F., Huang, S., Pelrine, K., and Rabbany, R., 2022. Towards better evaluation for dynamic link prediction. S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, eds. *Advances in neural information processing systems* [Online]. Vol. 35. Curran Associates, Inc., pp.32928–32941. Available from: https://proceedings.neurips.cc/paper_files/paper/2022/file/d49042a5d49818711c401d34172f9900-Paper-Datasets_and_Benchmarks.pdf.

Pytorch rnn documentation, 2024. https://pypi.org/project/py-tgb/. [Accessed 05-09-2024].

Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., and Bronstein, M., 2020. Temporal graph networks for deep learning on dynamic graphs. *Icml 2020 workshop on graph representation learning*.

Russell, S. and Norvig, P., 2022. Model Selection and Optimization. *Artificial intelligence: a modern approach*. Fourth Edition. Pearson Education, pp.665–672.

Sankar, A., Wu, Y., Gou, L., Zhang, W., and Yang, H., 2020. Dysat: deep neural representation learning on dynamic graphs via self-attention networks. *Proceedings of the 13th international conference on web search and data mining* [Online], Wsdm '20. Houston, TX, USA: Association for Computing Machinery, pp.519–527. Available from: https://doi.org/10.1145/3336191.3371845.

Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., and Monfardini, G., 2009. The graph neural network model. *Ieee transactions on neural networks* [Online], 20(1), pp.61–80. Available from: https://doi.org/10.1109/TNN.2008.2005605.

Song, W., Xiao, Z., Wang, Y., Charlin, L., Zhang, M., and Tang, J., 2019. Session-based social recommendation via dynamic graph attention networks. *Proceedings of the twelfth acm international conference on web search and data mining* [Online], Wsdm '19. Melbourne VIC, Australia: Association for Computing Machinery, pp.555–563. Available from: https://doi.org/10.1145/3289600.3290989.

Tolstikhin, I.O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., Lucic, M., and Dosovitskiy, A., 2021. Mlp-mixer: an all-mlp architecture for vision. M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J.W. Vaughan, eds. *Advances in neural information processing systems* [Online]. Vol. 34. Curran Associates, Inc., pp.24261–24272. Available from: https://proceedings.neurips.cc/paper_files/paper/2021/file/cba0a4ee5ccd02fda0fe3f9a3e7b89fe-Paper.pdf.

Trivedi, R., Farajtabar, M., Biswal, P., and Zha, H., 2019. Dyrep: learning representations over dynamic graphs. *International conference on learning representations* [Online]. Available from: https://openreview.net/forum?id=HyePrhR5KX.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, and Polosukhin, I., 2017. Attention is all you need. *Advances in neural information processing systems* [Online]. Vol. 30. Curran Associates, Inc. Available from: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y., 2018. Graph Attention Networks. *International conference on learning representations* [Online]. accepted as poster. Available from: https://openreview.net/forum?id=rJXMpikCZ.

Wang, Y., Chang, Y.-Y., Liu, Y., Leskovec, J., and Li, P., 2021. Inductive representation learning in temporal networks via causal anonymous walks. *International conference on learning representations* [Online]. Available from: https://openreview.net/forum?id=KYPz4YsCPj.

Wang, Y., Zhao, Y., Zhang, Y., and Derr, T., 2023. Collaboration-aware graph convolutional network for recommender systems [Online], Www '23. <conf-loc>, <city>Austin</city>, <state>TX</state>, <country>USA</country>, </conf-loc>: Association for Computing Machinery, pp.91–101. Available from: https://doi.org/10.1145/3543507.3583229.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., Platen, P. von, Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., and Rush, A., 2020. Transformers: state-of-the-art natural language processing. Q. Liu and D. Schlangen, eds. *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations* [Online]. Online: Association for Computational Linguistics, pp.38–45. Available from: https://doi.org/10.18653/v1/2020.emnlp-demos.6.

Wu, S., Sun, F., Zhang, W., Xie, X., and Cui, B., 2022. Graph neural networks in recommender systems: a survey. *Acm comput. surv.* [Online], 55(5). Available from: https://doi.org/10.1145/3535101.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P.S., 2021. A comprehensive survey on graph neural networks. *Ieee transactions on neural networks and learning systems* [Online], 32(1), pp.4–24. Available from: https://doi.org/10.1109/tnnls.2020.2978386.

Xu, da, ruan, chuanwei, korpeoglu, evren, kumar, sushant, and achan, kannan, 2020. Inductive representation learning on temporal graphs. *International conference on learning representations (iclr)*.

Yu, B., Yin, H., and Zhu, Z., 2018. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. *Proceedings of the 27th international joint conference on artificial intelligence*, Ijcai'18. Stockholm, Sweden: AAAI Press, pp.3634–3640.

Yu, L., Sun, L., Du, B., and Lv, W., 2023. Towards better dynamic graph learning: new architecture and unified library. A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, eds. *Advances in neural information processing systems* [Online]. Vol. 36. Curran Associates, Inc., pp.67686–67700. Available from: https://proceedings.neurips.cc/paper_files/paper/2023/file/d611019afba70d547bd595e8a4158f55-Paper-Conference.pdf.

Zhu, L., Guo, D., Yin, J., Steeg, G.V., and Galstyan, A., 2016. Scalable temporal latent space inference for link prediction in dynamic social networks. *Ieee transactions on knowledge and data engineering* [Online], 28(10), pp.2765–2777. Available from: https://doi.org/10.1109/TKDE.2016.2591009.

List of Figures

List of Tables