Degree Project in Computer Science and Engineering

Second cycle, 30 credits

Lightweight Memory Networks for Link Prediction

Generalization of the LiMNet Architecture

TITOUAN MAZIER

Lightweight Memory Networks for Link Prediction

Generalization of the LiMNet Architecture

TITOUAN MAZIER

Master's Program, Computer Science
Date: June 05, 2025

Supervisors: Šarūnas Girdzijauskas and Lodovico Giaretta
Examiner: Viktoria Fodor
School of Electrical Engineering and Computer Science
Host company: RISE, Research Institute of Sweden
Swedish title: Lightweight Memory Networks för Länkprediktion
Swedish subtitle: Generalization av LiMNet Arkitektur

Abstract

User-item recommendation is a central challenge for search engines, social media platforms, and streaming services, due to the need to model both relational structures and temporal dynamics. Many existing solutions address these two aspects separately, limiting their ability to fully capture user behavior.

In this work, we attempt to bridge that gap by evaluating Lightweight Memory Networks (LiMNet), a model designed to preserve causal relationships within sequences of temporal interactions. To assess its potential, we developed a benchmarking framework for user-item interaction prediction. We compared LiMNet against Jodie, a state-of-the-art baseline, across three real-world datasets: Wikipedia page edits, Reddit post submissions, and LastFM music streams. These datasets vary in scale and interaction patterns, providing a comprehensive testbed.

Our results show that while LiMNet offers advantages in efficiency and adaptability, it consistently underperforms compared to Jodie in predictive accuracy. Additionally, our findings hint at a consistent bias across all datasets toward short-term global popularity. This suggests that existing models may be overfitting to recent trends rather than learning long-term user preferences, highlighting a potential limitation in the current evaluation paradigms.

Keywords

Graph Representation Learning, Temporal Interaction Networks, Link Prediction, Data Mining, Machine Learning, Recommendations

Sammanfattning

If you are writing your thesis in English, you can leave this until the draft version that goes to your opponent for the written opposition. In this way, you can provide the English and Swedish abstract/summary information that can be used in the announcement for your oral presentation. If you are writing your thesis in English, then this section can be a summary targeted at a more general reader. However, if you are writing your thesis in Swedish, then the reverse is true – your abstract should be for your target audience, while an English summary can be written targeted at a more general audience. This means that the English abstract and Swedish sammnfattning or Swedish abstract and English summary need not be literal translations of each other.

The abstract in the language used for the thesis should be the first abstract, while the Summary/Sammanfattning in the other language can follow

Nyckelord

Graph Representation Learning, Temporal Interaction Networks, Link Prediction, Data Mining, Machine Learning, Recommendations

Acknowledgments

It is nice to acknowledge the people that have helped you. It is also necessary to acknowledge any special permissions that you have gotten – for example, getting permission from the copyright owner to reproducea figure. In this case, you should acknowledge them and this permission here and in the figure's caption. Note: If you do not have the copyright owner's permission, then you cannot use any copyrighted figures/tables/…. Unless stated otherwise all figures/tables/…are generally copyrighted.

I detta kapitel kan du ev nämna något om din bakgrund om det påverkar rapporten på något sätt. Har du t ex inte möjlighet att skriva perfekt svenska för att du är nyanländ till landet kan det vara på sin plats att nämna detta här. OBS, detta får dock inte vara en ursäkt för att lämna in en rapport med undermåligt språk, undermålig grammatik och stavning (t ex får fel som en automatisk stavningskontroll och grammatikkontroll kan upptäcka inte förekomma) En dualism som måste hanteras i hela rapporten och projektet

Stockholm, June 2025
Titouan Mazier

Contents

1. Introduction 1
1.1. Motivation 1
1.2. Problem 2
1.3. Limitations 3
1.4. Contributions 3

2. Background 5
2.1. User-Item Link Prediction 5
2.2. Graph Representation Learning 6
2.3. Dynamic Graphs 7
2.4. Cross-RNN 8
2.5. LiMNet 9

3. Method 11
3.1. Datasets 11
3.2. Experimental framework 12
3.2.1. Data preparation 13
3.2.2. Batching Strategy 13
3.2.3. Evaluation and Training loop 14
3.2.4. Embedding comparison 15
3.2.5. Code Reusability 16
3.3. Adaptations of the Lightweight Memory Networks (LiMNet) architecture 17
3.3.1. Loss functions 17
3.3.2. Addition of time features 18
3.3.3. Embedding normalization 18
3.3.4. Stacking of multiple LiMNet layers 19
3.4. Baseline 19

4. Experiments 21
4.1. Improvements on limnet 22
4.1.1. Adding time features 22
4.1.2. Normalizing the embeddings 22
4.1.3. Stacking layers 23
4.2. Impact of the sequence size on the results 24

5. Conclusions 27
5.1. Limitations 27
5.2. Future Works 27

5.3. Ethics and Sustainability 28
References 29

List of Figures

Figure 2.1 Architecture of LiMNet. 10
Figure 3.1 Schema of the evaluation framework. Blue indicates that the implementation is tight to the model evaluated. 14
Figure 3.2 Architecture of LiMNet with two layers. 19
Figure 4.1 Comparison of LiMNet and Jodie models. 21
Figure 4.2 Performances of the LiMNet model with time features added. 22
Figure 4.3 Performances of the LiMNet model with and without normalization of the embeddings. 23
Figure 4.4 Performances of the LiMNet model with various numbers of stacked layers. 23
Figure 4.5 Effect of the sequence length on the models' performances. . . 24

List of Tables

Table 3.1 Details of the datasets. 12

Acronyms and Abbreviations

CTDG – **Continuous-Time Dynamic Graph**: Dynamic graph were each edge is associated with a timestamp. 7

GRL – **Graph Representation Learning**: The task of computing high level representation of graphs, subgraphs, nodes or edges. Used as inputs for Machine Learning algorithms. 6, 7

GRU – **Gated Recurrent Unit**: A type of recurrent neural network that simplifies the LSTM architecture by combining the forget and input gates into a single update gate. 9

LiMNet – **Lightweight Memory Networks**: A neural network architecture that focuses on efficient memory utilization and lightweight design for temporal interaction network embeddings computations. vii, ix, 2, 3, 9, 10, 11, 13, 17, 18, 19, 20, 21, 22, 23, 24, 27, 28

LSTM – **Long-Short Term Memory**: A type of recurrent neural network designed to effectively learn and remember short and long-term dependencies in sequential data by using specialized memory cells and gating mechanisms. 9

MRR – **Mean Reciprocal Rank**: Metric used to compute the success of a ranking algorithm. It is a value comprised between 0 and 1 where 0 indicates a total absence of the expected result in the ranked items and 1 mean that the expected item is always ranked first. 22, 25

RNN – **Recurrent Neural Network**: A neural network whose output depends on both the inputs and on the state of an internal memory that is updated with each input. 8, 9

Chapter 1
Introduction

1.1. Motivation

The rapid expansion of digital technologies has led to an overwhelming abundance of information, making it increasingly difficult to identify relevant and meaningful content [1], [2]. In response to this challenge, search engines and recommendation systems have become essential tools for filtering and navigating vast data landscapes [3].

Both systems share a common objective: to deliver information that is most relevant to the user. Content personalization has emerged as a particularly effective approach among the many techniques developed to achieve this. Rather than applying a one-size-fits-all filter, personalized systems adapt their output based on user-specific context such as search history, demographics, and past interactions.

Content personalization lies at the heart of recommendation systems and is highly effective in enhancing search engine performance. For instance, a search query for the term “football” should yield different results for a user interested in American football compared to someone seeking information about association football (soccer).

Content personalization can be represented as an algorithm that takes user-specific information as input and produces a ranked list of items from a larger catalog. These items can include any kind of content available to the user, such as songs in a music streaming service or web pages returned by a search engine.

Evaluating the performance of such algorithms typically requires knowledge of which items are genuinely relevant to each user. However, obtaining this information can be costly and, in some cases, unfeasible. On the other hand, user behaviors—such as clicks, listens, or edits—are easy to collect at scale. As a result, content recommendation is often approximated as an interaction prediction task, where the goal is to predict future user-item interactions based on past behavior.

Interaction prediction is a self-supervised learning task in which past user-item interactions serve first as labels and later as input features used to predict future interactions. Unlike other self-supervised tasks, it is distinguished by the inherently relational nature of the data: each interaction connects users and items, forming a web of dependencies that influence future behavior. As a result, these interactions are often modeled as user-item networks to highlight the structural relationships between entities.

Besides, the temporal dimension, which includes the order and timing of interactions, typically plays a critical role in accurately modeling and understanding user behavior over time. One natural way to model both temporal and relational information is to use a temporal interaction network, that is, a network where each interaction is linked to a timestamp. This way, following the order of the timestamps reveals the network and it's evolutions, presenting an ever changing map of relationships between the users and the items and allowing for more nuanced and dynamic predictions. Yet, few solutions rely on this model, despite it's intuitive definition, which motivated this work to look further into existing solutions.

LiMNet [4] is a simple machine learning model designed to process temporal interactions in a causal manner, leveraging both relational information and the order of interactions. The model has demonstrated strong performance in tasks such as botnet detection in IoT networks, and it is built in a modular, adaptive way that makes it easy to apply to other problems. Given these promising characteristics and its ability to exploit precisely the types of information that make interaction prediction challenging, LiMNet appears to be a compelling candidate for this task.

1.2. Problem

The core research question for this work is the following:

"Does LiMNet have the potential to compete against state-of-the-art models on the task of interaction prediction for user-item network?"

LiMNet has shown significant promise, yet its evaluation has been limited to botnet detection in IoT networks [4], with some additional results available for cryptocurrency fraud detection [5]. This project aims to apply and assess LiMNet in the context of interaction prediction, thereby evaluating the model's generalizability across a broader range of tasks. Addressing this research question supports two main goals: first, to explore the applicability

of the LiMNet architecture to diverse domains, and second, to better understand the characteristics that drive the success of similar state-of-the-art interaction prediction models.

To this end, we develop a flexible evaluation framework supporting multiple models while ensuring fair and consistent comparisons. Additionally, we implement and evaluate several architectural adaptations to LiMNet , aiming not only to improve its performance on interaction prediction but also to further investigate the design space enabled by its modular structure.

1.3. Limitations

One alternative approach to interaction prediction involves building a classifier to determine whether a given user is likely to interact with a specific item. This approach is excluded from the present work, which instead focuses on computing embeddings and generating ranked item lists.

Another limitation of this study is its strict focus on user-item networks. Interactions between entities of the same type, such as user-user or item-item connections, are not considered. For instance, tasks like friendship prediction on a social network fall outside the scope of this project. Consequently, each entity is strictly classified as either a user or an item, with no allowance for hybrid or hierarchical roles. And only one kind of interaction is considered, e.g. user listening to a song but not artist posting a song.

Furthermore, this work considers only punctual interactions, i.e. discrete events occurring at specific points in time. Continuous interactions are excluded, and due to data limitations, the duration of interactions is not modeled. Although interaction durations could offer meaningful insights into engagement or relevance, they remain outside the scope of this thesis.

1.4. Contributions

The main contributions of this project are as follows:

- Development and publication of a framework for evaluating models on the user-item interaction prediction task.
- Presentation and evaluation of LiMNet, an embedding model originally proposed for IoT botnet detection. Our evaluation shows that, in its current form, LiMNet does not match the performance of state-of-the-art baselines.

- Proposal and testing of architectural modifications to LiMNet aimed at improving its performance in the context of user-item interaction prediction.
- Reproduction and evaluation of Jodie, a state-of-the-art model for temporal interaction prediction. Interestingly, our implementation produced results that exceeded expectations.
- Identification of a possible structural bias in the benchmark datasets, suggesting they may favor global popularity trends over more complex, long-term behavioral patterns.

Chapter 2
Background

This chapter provides the background for the project. In Section 2.1, we provide an overview of link prediction and the classical solutions for the problem. In Section 2.2, we further develop the concept of graph embedding and some common methods to create them. Then, in Section 2.3, we discuss the addition of a time dimension in graph-shaped data and the way it can be exploited, followed by a presentation of cross-RNN architectures in Section 2.4. Finally, we present the model of interest for this work in Section 2.5 and why we believe that it is a relevant addition to the task of link prediction.

2.1. User-Item Link Prediction

Interaction prediction can often be formulated as a link prediction problem within a user-item graph. In such a graph, each user and each item is represented as a node. Each user interaction with an item is registered as an edge in the graph. Predicting future interactions comes down to predicting which edges are likely to appear next.

In this project, we focus specifically on user-item interaction networks—systems in which all interactions occur between distinct user and item entities. Users and items play fundamentally different roles: users initiate interactions, while items are the targets. This introduces a structural asymmetry in the network, where interactions are always directed from a user to an item. In practice, this means that prediction tasks are framed from the user's perspective, with the goal of identifying the most relevant items they are likely to interact with next.

There are two primary approaches to this problem: one based on graph analytics and the other on feature-based methods. Graph analytics focuses on measuring the proximity between a user and various items in the graph, leveraging insights from the user's past interactions and the behaviors of similar users. For instance, if two users have listened to the same set of songs, one may likely enjoy the songs the other has listened to. Graph theory offers a variety of methods to compute closeness between nodes, including shortest

path lengths, the number of shared neighbors, or the exclusivity of those shared neighbors.

The second approach leverages additional information beyond the user-item relationship. Most real-world systems provide rich metadata about interactions, users, and items. For example, in a music streaming service, a song may include attributes like genre and duration, while a user may be characterized by age or selected language. These attributes are referred to as features, and utilizing them is central to machine learning approaches. Unlike graph-based methods, feature-based models recommend items by identifying similar users based on shared characteristics. Continuing the music example, the system might learn that songs with lyrics in Swedish are less likely to appeal to users that don't use Swedish as their primary language.

The challenge lies in integrating both approaches. Lichtenwalter et al. proposed framing link prediction as a supervised machine learning task, where the objective is to predict whether an edge will form between a given user-item pair in the future. To incorporate relational data into the model, graph closeness metrics are added to the user and item features[6]. This setup uses graph structure not as the direct basis for prediction but as a source of enriched features, enabling machine learning algorithms to work with abstracted representations of the graph.

Despite their strengths, these methods share a common drawback: for each user, a score must be computed for every possible item to generate a recommendation. This becomes computationally infeasible when dealing with large item catalogs. A common solution is to learn high-dimensional embeddings for users and items separately and then compute a similarity score, such as a dot product or distance measure, to rank items. This transforms the problem into a nearest-neighbor search, a well-studied task with many efficient and scalable solutions.

2.2. Graph Representation Learning

The task of learning high-level representations from graph data is known as Graph Representation Learning (GRL) . GRL encompasses a broad family of machine learning techniques aimed at transforming the complex, non-Euclidean structure of graphs into low-dimensional Euclidean representations

(i.e., numerical vectors), making them suitable for use in downstream machine learning tasks.

GRL methods can be applied to a wide range of problems. These include classifying entire graph structures, such as molecular graphs; extracting subgraph representations from knowledge graphs to be used in large language models; and, most commonly, generating node embeddings. These node embeddings must encode not only the intrinsic features of individual nodes but also the context in which they appear. This context typically includes neighboring nodes and their corresponding features and positions within the graph.

2.3. Dynamic Graphs

Much of the information generated in real-world networks is inherently dynamic, particularly in user-item interaction networks, where each interaction occurs at a specific point in time. Despite this, the temporal dimension is often ignored or simplified in order to reduce modeling complexity. Yet, temporal information carries unique value, enabling models to capture not just patterns but also the causal relationships between interactions.

Causality refers to the principle that actions can influence future outcomes. This is especially important when studying processes that propagate through networks, such as the spread of information, trends, or behaviors. In these scenarios, an interaction may change the state of the involved nodes, making it necessary to treat the same node differently depending on the time of observation. While humans intuitively understand these evolving patterns, many standard GRL techniques disregard temporal order, propagating information across the graph without regard to when interactions occurred.

In their review of dynamic networks [7], Zheng et al. outline two common approaches to incorporating temporal information into two models. The first involves representing the graph as a sequence of discrete snapshots, each corresponding to a specific time step. The second, known as Continuous-Time Dynamic Graph (CTDG) , treats each graph update as an event timestamped in continuous time, typically the addition or removal of an edge.

This work focuses on CTDG , where all events are modeled as punctual interactions, also referred to as temporal interaction networks. These networks

offer a faithful representation of many real-world systems, as they continuously track changes over time. However, they come with a challenge: the underlying graph structure becomes ephemeral, with each edge appearing only momentarily. As a result, these networks are often better understood as interaction streams rather than static or evolving graphs with persistent edges.

A popular approach for leveraging temporal data when generating node embeddings is to maintain a memory of embeddings and update them as interactions occur. One foundational model in this domain is DeepCoevolve [8], a model for link prediction that uses two components: a cross-RNN (further detailed in Section 2.4) to update user and item representations, and an intensity function to predict the likelihood of future interactions at any given time. Following DeepCoevolve, several cross-RNN-based models have been proposed, achieving notable performance improvements.

JODIE [9] extends DeepCoevolve by introducing a static embedding component alongside the dynamic one. The cross-RNN tracks the trajectory of users and items over time, while a neural projection layer predicts their future embeddings at different time steps, replacing the intensity function used in DeepCoevolve.

DeePRed [10] builds upon DeepCoevolve with the goal of simplifying and accelerating training by removing recurrence from the cross-RNN mechanism. Instead, dynamic embeddings are computed directly from static embeddings, avoiding recursive updates. The absence of long-term memory is addressed through the use of a sliding context window and an attention mechanism that identifies and weights the most relevant past interactions.

2.4. Cross-RNN

The key mechanism underlying the models discussed in the previous section is known as cross-RNN, where RNN stands for Recurrent Neural Network. A RNN is a type of neural network designed to process sequential data by maintaining and updating a memory state across time steps. Formally, a RNN layer is defined as:

$$\boldsymbol{o}(\boldsymbol{i}_t) = f(\boldsymbol{i}_t, \boldsymbol{h}_{t-1}) \tag{1}$$

$$\boldsymbol{h}_t = g(\boldsymbol{i}_t, \boldsymbol{h}_{t-1}) \tag{2}$$

Here, t denotes the time step of the input $\boldsymbol{i}_t$. The function $\boldsymbol{o}(\boldsymbol{i}_t)$ produces the layer's output, and $\boldsymbol{h}_t$ represents the updated memory after processing $\boldsymbol{i}_t$. The functions f and g are parameterized transformations, typically involving learned weights. Popular RNN architectures, such as Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) , are designed to retain long-term dependencies more effectively than simple RNNs . LSTM maintains two types of memory (short-term and long-term), whereas GRU simplifies this structure by using a single memory vector with a gating mechanism. In practice, GRUs achieve performance comparable to LSTMs while being computationally less demanding [11].

A cross-RNN layer extends this concept by maintaining separate memory states for all nodes in a graph. At time t, the memory is represented as: $\boldsymbol{H}_t = (\boldsymbol{h}_t^u)_{u \in \mathbb{U}} \cup (\boldsymbol{h}_t^i)_{i \in \mathbb{I}}$ where $\mathbb{U}$ and $\mathbb{I}$ denote the sets of users and items, respectively. For each interaction $(u, i, t, \boldsymbol{f})$, the memory states of the involved user u and item i are updated according to:

$$\boldsymbol{h}_t^u = g^u(\boldsymbol{h}_{t-1}^u, \boldsymbol{h}_{t-1}^i, t, \boldsymbol{f}) \tag{3}$$

$$\boldsymbol{h}_t^i = g^i(\boldsymbol{h}_{t-1}^i, \boldsymbol{h}_{t-1}^u, t, \boldsymbol{f}) \tag{4}$$

For all other nodes $v \in (\mathbb{U} \setminus \{u\}) \cup (\mathbb{I} \setminus \{i\})$, the memory remains unchanged.

Here, g^u and g^i are node-type-specific update functions analogous to g in a standard RNN . LSTM and GRU cells can be used within the cross-RNN framework, with the key distinction that memory management is handled externally to the cell.

The primary advantage of cross-RNN architectures is their inherent preservation of causality. Since updates depend strictly on past states, the model respects the temporal order of interactions by design. However, this sequential nature introduces a limitation: cross-RNN models cannot be parallelized over the sequence during training. Fortunately, this constraint primarily affects training efficiency, during inference, each interaction can be processed independently, making the approach practical for real-time applications.

2.5. LiMNet

LiMNet is a cross-RNN model designed to optimize memory utilization and computational efficiency during inference. In its original formulation [4],

LiMNet is part of a comprehensive framework for botnet detection in IoT networks. The framework includes four main components: an input feature map, a generalization layer, an output feature map, and a response layer. For the purposes of this work, however, we consider only the generalization layer. Since the other components are task-specific, the term "LiMNet" in this thesis refers exclusively to the generalization layer.

As a graph embedding module, LiMNet provides a straightforward implementation of a cross-RNN mechanism, Figure 2.1 proposes a visualization of its architecture. This simplicity offers two main advantages. First, LiMNet is highly efficient at inference time, with memory requirements that scale linearly with the number of nodes in the network. Second, it offers strong flexibility in handling dynamic node sets: when a new node is introduced, its embedding can be computed immediately without retraining the model. Deleting a node is even simpler: its embedding can just be removed from memory without consequences for the other embeddings.

LiMNet has already demonstrated promising results in tasks such as IoT botnet detection [4] and cryptocurrency fraud detection [5]. Given its design and prior success, it is a compelling candidate for link prediction tasks, particularly in settings where computational efficiency is a key constraint.

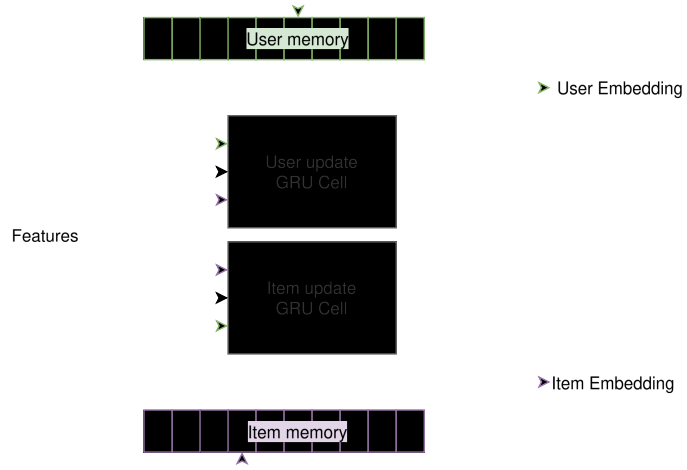

Figure 2.1: Architecture of LiMNet.

Chapter 3
Method

In this chapter, we detail the experiments conducted throughout this work. First in Section 3.1, we introduce the datasets used in this work. Section 3.2 details the framework developed to conduct the experiments in a fair and controlled environment. Next, we present in Section 3.3 the various adaptations proposed for LiMNet to solve the task of link-prediction. Finally, we discuss in Section 3.4 exploration we conducted with the baselines.

3.1. Datasets

This project uses three publicly available datasets sourced from the Stanford Large Network Dataset Collection (accessible at snap.stanford.edu/jodie/#datasets). These datasets were originally compiled by Kumar et al. [9] and have since become widely adopted as de facto standard benchmarks for evaluating interaction prediction models.

- **Wikipedia edits:** This dataset captures edits made to Wikipedia pages over the course of one month. It includes the 1,000 most edited pages during that period and 8,227 users who each made at least five edits to these pages. In total, it contains 157,474 interactions.

- **Reddit posts:** Built using a similar methodology as the Wikipedia dataset, this dataset records 672,447 posts made by the 10,000 most active users on the 1,000 most active subreddits within a month.

- **LastFM songs listens:** This dataset logs 1,293,103 music streams performed by 1,000 users on the 1,000 most listened-to songs on the LastFM platform, again over the span of one month.

The original dataset publication also included a fourth dataset containing interactions between 7,047 students and 97 courses on a MOOC platform. However, we excluded this dataset from our experiments, as it does not reflect a relevant use case for interaction prediction. Users on MOOC platforms typically have a clear intent when accessing the platform, which diminishes the predictive value of interaction modeling in this context.

	Wikipedia	Reddit	LastFM
Users	8,227	10,000	1,000
Items	1,000	1,000	1,000
Interactions	157,474	672,447	1,293,103
Unique edges	18,257	78,516	154,993

Table 3.1: Details of the datasets.

Table 3.1 summarizes the key characteristics of the three selected datasets. While Wikipedia and Reddit have comparable numbers of users and items, Reddit is significantly denser, with roughly four times more interactions. LastFM is denser still, with nearly 20 times the interaction density of Reddit. Notably, LastFM also exhibits a perfectly balanced user-to-item ratio, in contrast to the other two datasets.

3.2. Experimental framework

Evaluating embedding models is inherently complex due to the variety of input and output formats, as well as the diversity of training and inference procedures. Despite these differences, a fair and consistent evaluation across models must be ensured.

This complexity is further amplified in the case of temporal graphs, which can be interpreted in multiple ways depending on the structural and temporal aspects one wishes to emphasize. A temporal graph may be decomposed into a sequence of static snapshots taken at regular intervals, represented as a continuous time series of events, or treated as a dynamic structure where nodes and edges evolve over time [7]. These different interpretations offer varied trade-offs in terms of temporal resolution, scalability, expressiveness, and design opportunities, without any single approach being universally optimal.

Our implementation is publicly available on GitHub at: https://github.com/mazerti/link-prediction.

The following subsections describe the design choices that guided the development of our evaluation framework. These are organized into four key components: data preparation, batching strategy, evaluation and training loop, and embedding comparison.

3.2.1. Data preparation

Each recorded interaction in the datasets provides three types of information: the identifiers of the interacting user and item, the timestamp of the interaction, and a set of optional features that offer additional context. In most cases, the evaluated models rely primarily on the user and item identifiers, along with the implicit temporal order of the interactions. As such, the framework consistently supplies user and item IDs in the exact sequence in which the interactions occur.

The framework also supports the inclusion of custom features in the input. These features can be specified either by the user through a configuration file or automatically requested by the model implementation during initialization. This flexibility ensures that models requiring specific features can be seamlessly integrated without manual intervention. Time-related features, in particular, benefit from this design. For example, some models use the time delta between successive interactions by the same user. While calculating this value at inference time would be computationally expensive, as it requires real-time tracking of each user's previous interactions, it can be efficiently pre-computed when the full interaction history is available, reducing it to a straightforward query operation.

3.2.2. Batching Strategy

Temporal interaction modeling inherently involves a tradeoff between preserving the sequential nature of data and maximizing training efficiency through parallelism. As highlighted in prior work [9], [10], maintaining causality often comes at the cost of reduced parallelization capabilities. In the JODIE model, Kumar et al. addressed this by designing a graph-structure-aware batching strategy that retains temporal coherence while enabling some degree of parallel processing [9]. Meanwhile, Kefato et al. proposed an alternative in DeePRed by eliminating recursion entirely, replacing dynamic embeddings with static approximations to simplify training [10].

Drawing inspiration from the original LiMNet framework [4], we adopt a different approach: slicing the dataset into fixed-size sequences. The rationale is that sufficiently long sequences can serve as a reasonable approximation of the full interaction history. Each individual sequence is processed in temporal order, thereby preserving internal causal structure. However, because sequences are independent of one another, they can be processed in parallel, significantly accelerating training.

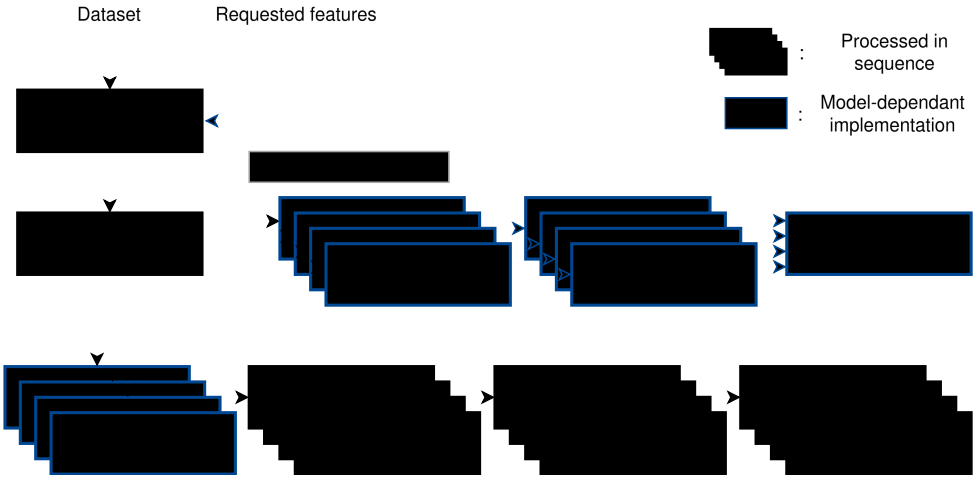

Figure 3.1: Schema of the evaluation framework. Blue indicates that the implementation is tight to the model evaluated.

This strategy offers a practical compromise. It enables the model to learn from temporally ordered data without incurring the full computational burden of processing the entire dataset sequentially. Moreover, it allows for flexibility in choosing the sequence length, which can be tuned to balance modeling capacity and computational efficiency. In our experiments, we found that shorter sequences often performed comparably well, suggesting that most relevant predictive signals are contained in recent interaction history.

3.2.3. Evaluation and Training loop

Designing a framework that accommodates any model for a given task is inherently challenging, as different models are often developed based on varying problem formulations. One major difference lies in the structure of the inputs. As discussed in Section 3.2.1, the framework addresses this by leveraging input features to bridge discrepancies between models. Another key distinction involves the nature of the outputs. Although all models ultimately aim to identify relevant items, they approach this goal in different ways. In the context of interaction prediction, predictions can be generated by computing embeddings to be compared, by directly scoring items, or by estimating the likelihood of future user-item interactions.

This framework solely addresses the model creating user and item embeddings, simplifying the evaluation process and allowing it to be performed independently of the model that generates them. However, all training and

loss evaluation logic is encapsulated within the model implementations. This approach allows for diverse optimization strategies, including loss functions that depend on a model's internal memory state rather than exclusively on its outputs. These design choices are reflected in the framework architecture diagram presented in Figure 3.1.

3.2.4. Embedding comparison

The final challenge in the implementation concerns the use of embeddings. While embeddings are created to condense and represent interaction information, they are not the system's ultimate objective. The actual goal is to rank items for a given user such that the item with which the user will interact appears as high as possible on the list.

There are several ways to translate embeddings into rankings. Since this is not the central focus of the current work, we adopt a straightforward approach: ranking item embeddings based on their proximity to the user embedding. To accommodate the diversity of models tested, the framework supports two proximity metrics.

The first is the dot product of normalized embeddings, which is equivalent to the cosine similarity between vectors:

$$\text{dot\_product\_score}(\boldsymbol{e}^{\text{user}}, \boldsymbol{e}^{\text{item}}) = \frac{\boldsymbol{e}^{\text{user}}}{\|\boldsymbol{e}^{\text{user}}\|} \cdot \frac{\boldsymbol{e}^{\text{item}}}{\|\boldsymbol{e}^{\text{item}}\|} \tag{5}$$

A higher dot-product score indicates that the item embedding is more closely aligned with the user embedding, and thus should be ranked higher.

The second metric is the L2 distance, a generalization of Euclidean distance to k-dimensional space:

$$\text{L2\_score}(\boldsymbol{e}^{\text{user}}, \boldsymbol{e}^{\text{item}}) = \sqrt[k]{\|\boldsymbol{e}^{\text{user}} - \boldsymbol{e}^{\text{item}}\|^k} \tag{6}$$

For this score, smaller values correspond to closer embeddings and are therefore ranked higher.

In all experiments conducted in this work, performance is measured using both scoring methods. The highest result obtained between the two is reported to ensure fair evaluation across models.

3.2.5. Code Reusability

In exploratory research projects like this one, writing the entire codebase from scratch can be advantageous. This approach eliminates the burden of dependency management, avoids the need to thoroughly understand legacy implementations, and frees the researcher from conforming to existing frameworks. Building from the ground up also allows for alternative perspectives on the task at hand and allows researchers to concentrate on challenges arising from novel aspects of the work. However, the ability to reproduce experiments and to reuse existing models, either as baselines or as foundations for further development, remains critically important, it would thus be inappropriate to write such a framework without having future researchers in mind.

To balance these needs, the framework has been designed with clarity, reproducibility, and extensibility in mind. Three principles have guided its development: comprehensive documentation, centralized state management, and a functional programming approach. Every function in the framework is systematically documented[1] to help future researchers quickly grasp the implementation, whether to reuse the code or replicate its behavior in a new context.

To further streamline usability, state management has been centralized in a single component: the Context class. This class acts as a unified store for all stateful elements of the framework, ensuring that any part of the system can access necessary state variables with minimal effort.

In addition, the framework adheres to a functional programming-inspired style, favoring pure functions wherever possible for their conceptual simplicity and consistency. While this approach enhances readability and modularity, certain components, most notably the PyTorch modules, had to follow an object-oriented structure due to the requirements of external libraries.

This balance between clean design and practical flexibility ensures that the framework is easy to understand for future experiments.

[1] Due to time constraints, the code quality deteriorated a bit during the last steps of the projects, leading to some undocumented functions.

3.3. Adaptations of the LiMNet architecture

The primary goal of this work is to evaluate the performance of the LiMNet model on the link prediction task. As discussed in Section 2.5, the original implementation of LiMNet includes input and output mapping layers, as well as a response layer specifically designed for IoT botnet detection. For our purposes, these components were removed to better align the model with the requirements of interaction prediction.

3.3.1. Loss functions

The loss function also required adaptation. Unlike botnet detection, link prediction is not a classification task, making the original cross-entropy loss unsuitable. Instead, we opted for a composite loss combining two components.

The first is an objective loss that minimizes the distance between the embeddings of interacting users and items. This can be computed either as the mean squared error between the embeddings of the interacting user and item or, when embeddings are normalized, as the squared difference between their dot product and 1 (more on this in Section 3.3.3).

However, the objective loss alone is insufficient to train the model effectively. Neural networks tend to converge to trivial solutions if not properly constrained; in this case, minimizing only the distance between embeddings would eventually cause all embeddings to collapse to the same value. To mitigate this, we introduce a regularization loss that promotes information retention by maximizing the distance between different users' embeddings and between different items' embeddings.

This regularization loss is computed as:

$$L_{\text{reg}} = \boldsymbol{U}\boldsymbol{U}^T + \boldsymbol{I}\boldsymbol{I}^T \tag{7}$$

where $\boldsymbol{U}$ and $\boldsymbol{I}$ are the matrices containing all user and item embeddings, respectively.

In addition to simplifying the architecture and adapting the loss function, we propose three modifications aimed at enhancing the model's performance: the addition of time features, embedding normalization, and the stacking of multiple LiMNet layers.

3.3.2. Addition of time features

While LiMNet leverages the order of interactions to propagate information in a causal manner, it does not incorporate actual timestamps when computing embeddings. We hypothesized that this omission could lead to a loss of valuable temporal information that might otherwise help in predicting the most relevant items. To test this assumption, we introduced time-based features aimed at capturing when each interaction occurred. Specifically, we sought to model cyclic behavioral patterns such as differences in user activity between weekdays and weekends, or between day and night.

However, our datasets only include relative timestamps, which obscure the exact timing of interactions. As a workaround, we approximated cyclic patterns by applying a frequency decomposition to the timestamps. We computed two features to represent temporal cycles:

$$\cos\left(\frac{2\pi t}{\Delta}\right), \sin\left(\frac{2\pi t}{\Delta}\right) \tag{8}$$

where t is the timestamp of the interaction, and Δ is the duration of the pattern to be captured (e.g., one day or one week) expressed in the timestamp's time unit. This representation aims to provide the model with a more learnable form of temporal information, as machine learning models often struggle to extract patterns from raw one-dimensional values.

3.3.3. Embedding normalization

An efficient approach to computing dot-product scores (see Eq. 5) is to normalize all embeddings onto the unit sphere. In our implementation, we extended this normalization to include the embeddings stored in LiMNet 's memory. By ensuring that inputs to the cross-RNN mechanism are also normalized, we encourage the model to encode information primarily through angular relationships rather than magnitude.

This normalization has additional benefits. While the regularization loss is designed to prevent all embeddings from converging to the same direction, it does not explicitly prevent them from collapsing to zero. Since embeddings with a magnitude of zero are still orthogonal, they technically satisfy the regularization condition. Maintaining normalization throughout the network guards against this collapse by constraining all embeddings to lie on the surface of the unit sphere.

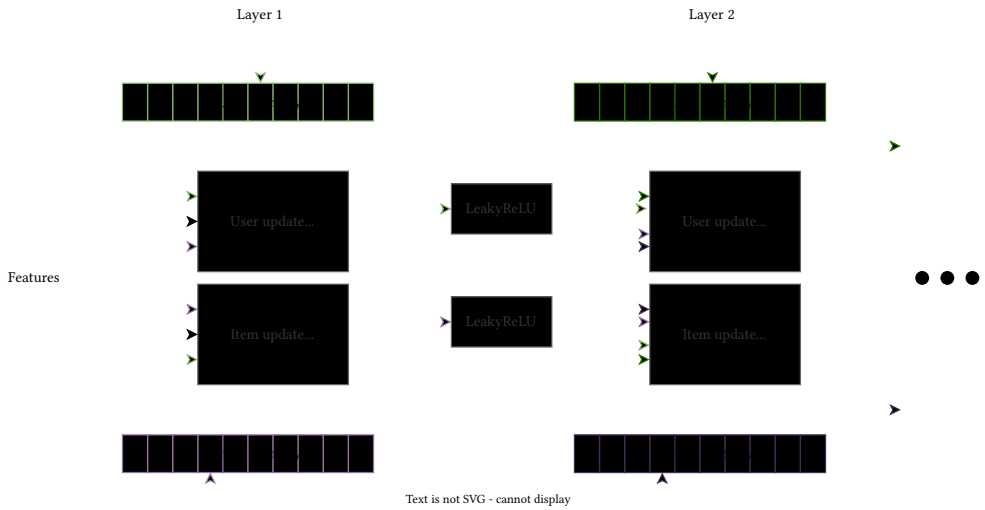

Figure 3.2: Architecture of LiMNet with two layers.

Our experiments showed a significant performance improvement with normalized embeddings. As a result, this modification was adopted as the default setting for all LiMNet -based experiments in this project.

3.3.4. Stacking of multiple LiMNet layers

The final adaptation we explored involved stacking multiple layers of the LiMNet architecture to form a deep recurrent neural network. This hierarchical design enhances the model's representational capacity by allowing it to learn progressively more abstract features across layers. Figure 3.2 illustrates the structure of the extended model.

To introduce non-linearity and further boost the expressiveness of the network, leaky ReLU activation functions were inserted between each pair of layers. This design choice aims to help the model capture more complex patterns in user-item interactions.

3.4. Baseline

We evaluated the performance of LiMNet against Jodie, a state-of-the-art cross-RNN model for learning embeddings in temporal interaction networks. We also attempted to implement DeePRed [10], but were unable to reproduce the performance reported in the original paper. As a result, DeePRed was excluded from our experimental comparisons.

Jodie, described in [9], shares the foundational use of cross-RNN embeddings with LiMNet but differs in three important ways. First, Jodie enhances its dynamic embeddings with one-hot representations of users and items to form the final embedding vectors. Second, it incorporates time deltas between consecutive user interactions via a projection mechanism designed to anticipate the trajectory of embeddings. Third, the model employs a specialized loss function to ensure that user and item embeddings do not shift too drastically in response to a single interaction.

Our implementation of Jodie differs from the original in two respects. We replaced the t-batch algorithm, used for creating training batches in the original paper, with fixed-length interaction sequences and omitted interaction features for the sake of simplicity. While Jodie does not require re-training to incorporate new interactions, it lacks LiMNet ‘s ability to dynamically handle the insertion or deletion of users and items.

These distinctions make Jodie a strong and informative baseline for evaluating LiMNet ‘s generalization to user-item interaction prediction tasks.

Chapter 4
Experiments

In this chapter, we present the experiments performed throughout this project. The first section covers how we tried to improve on the core of the limnet model. Then we evaluate the impact of the temporal information on the models.

To give some context, Figure 4.1 shows the performance of the two models we tested, LiMNet and Jodie. It stands clearly that Jodie is outperforming LiMNet by a wide margin, regardless of the dataset used.

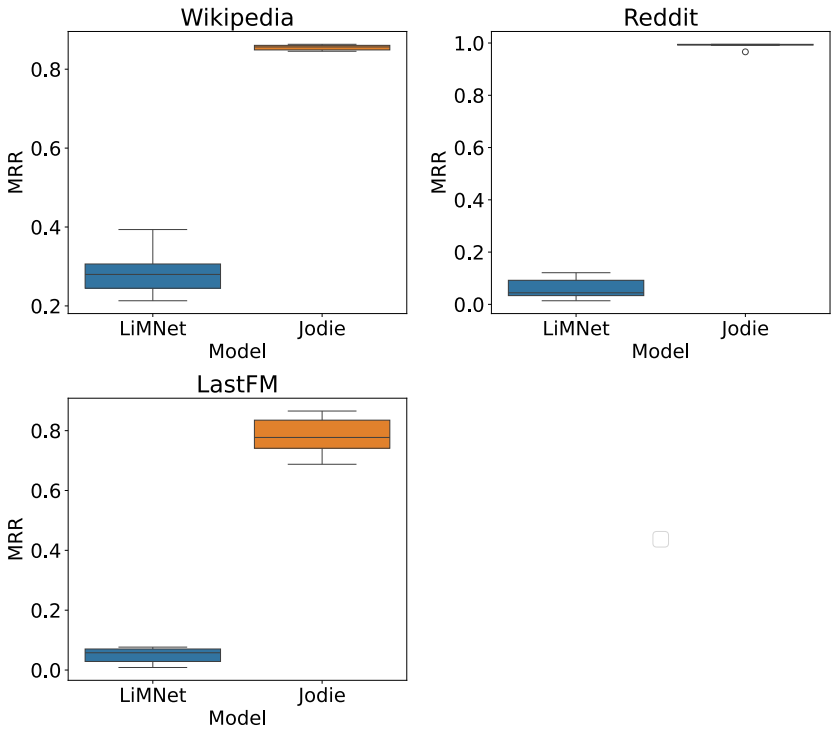

Figure 4.1: Comparison of LiMNet and Jodie models.

4.1. Improvements on limnet

Three different modifications have been tried with the hope of reducing the gap between LiMNet and Jodie. The three following subsections present these modifications and their experimental results.

4.1.1. Adding time features

The most important specificity of the models tested lies in their ability to leverage temporal relationships between users and items. However, LiMNet only exploits the order in which interactions happen, with no consideration for their exact time.

To resolve this absence, this experiment tries to pass the exact interaction timestamps as features to the LiMNet model. More specifically, the features passed to the model seek to capture cyclic patterns in the interactions. Unfortunately, the datasets only provide relative timestamps that do not carry the precise time and day of the interactions. We thus approximated these patterns through a frequency decomposition of the following form:

$$\cos\left(\frac{2\pi}{\Delta}t\right), \sin\left(\frac{2\pi}{\Delta}t\right) \tag{9}$$

Where t is the timestamp of the interaction, and Δ is the duration of the pattern we want to capture (a day or a week) in the unit of the timestamps. Using cosine and sine is a trick that provides the model with easily comparable features compared to directly passing the timestamps as a feature.

Figure 4.2 Shows the impact of these features on the measured Mean Reciprocal Rank (MRR) . Adding features seems to reduce the variance of the model, but may have a negative impact on the average results. In any case, it seems clear that the model struggles to deal with these additional features in a meaningful way. This experiment thus constitutes a case against using such features as a way to enhance the LiMNet model for link prediction.

4.1.2. Normalizing the embeddings

Forcing the embeddings to lie on the unit sphere pushes the model to encode information through the angle the embeddings form with the space origin rather than through their amplitude. This is appropriate when optimizing the embeddings for the dot-product score Eq. 5. In this experiment, we tried to systematically normalize the embeddings after each interaction; this

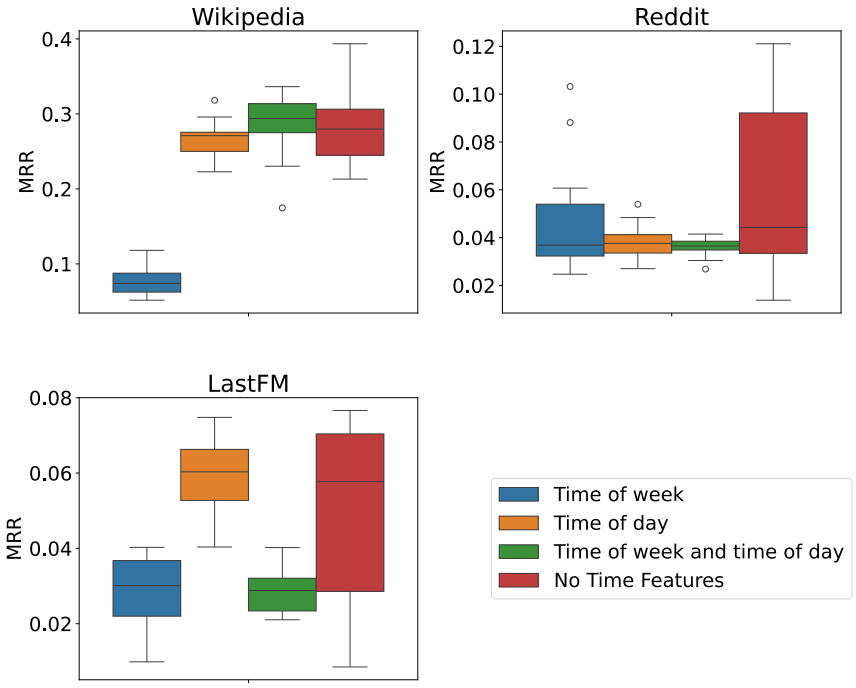

Figure 4.2: Performances of the LiMNet model with time features added.

way, not only the outputs but also the inputs of the cross-RNN are always normalized.

As you can see in Figure 4.3, this modification yields significantly better results. Thus, we apply it by default to all the other experiments presented in this report.

4.1.3. Stacking layers

This experiment tests whether stacking several layers of LiMNet could improve its results. Figure 3.2 illustrates the architecture of a stacked LiMNet model. In between layers, Leaky ReLU units have been added to add non-linearity and expressiveness to the model.

As shown in Figure 4.4, stacking more than two layers doesn't prove to increase the model performance. Another observation that can be made is that changing from one layer to two have a positive impact on the more

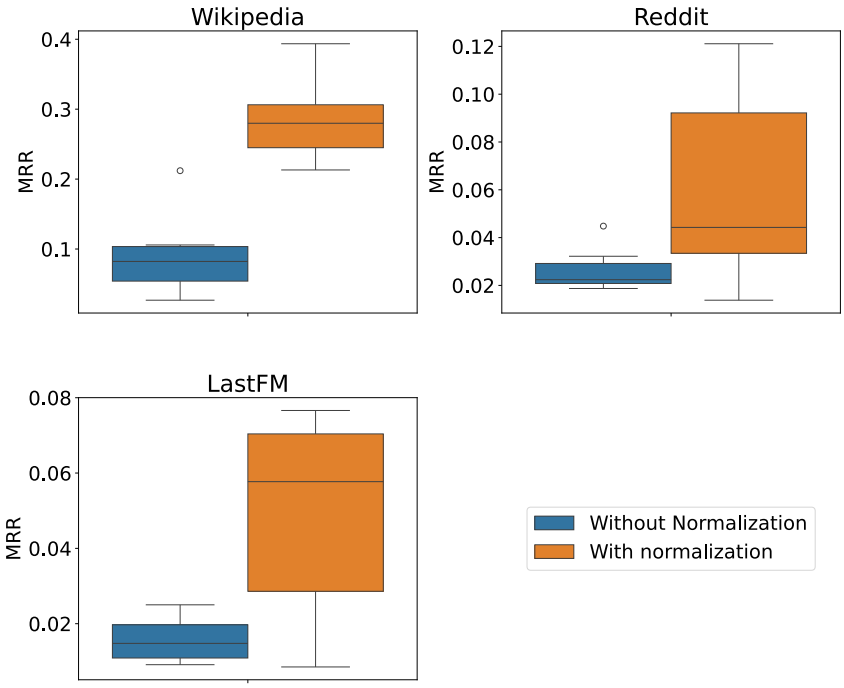

Figure 4.3: Performances of the LiMNet model with and without normalization of the embeddings.

complex dataset but tend to decrease the performance of the model for the simpler Wikipedia dataset.

4.2. Impact of the sequence size on the results

The last experiment aims to understand the impact of the temporal and sequential information on the performance of the models. It was performed by training and evaluating the models with sequence lengths of 16, 64, 256, and 1024, so that the model would only have access to up to this many successive interactions to perform their prediction.

The results displayed in Figure 4.5 show that both models perform, at best, as good with a bigger sequence length, which suggests a poor ability to exploit long-term information. However, except for LiMNet on Wikipedia, the sequence length seems to actually play only a marginal role in the

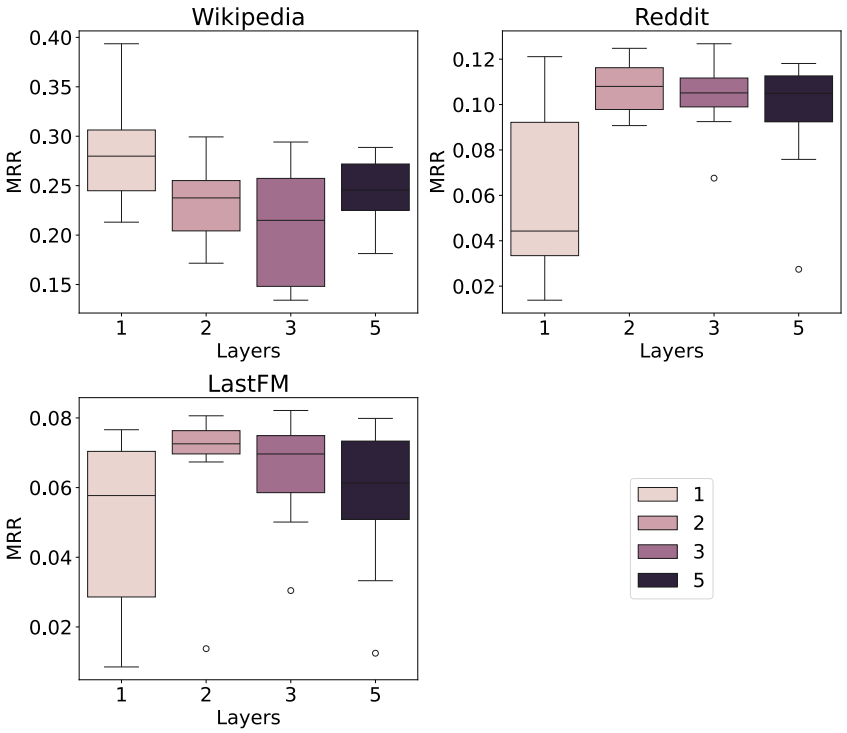

Figure 4.4: Performances of the LiMNet model with various numbers of stacked layers.

performance of both models, which suggests that they may not even alleviate temporal information as they were designed for.

Under the light of these observations, we assume that the models may have learned over short-term global popularity patterns instead of long-term local preferences. This behavior seems to make sense with regard to the nature of the datasets used, where recent interactions by other users are likely to provoke new interactions from other users. In addition, all these datasets contain only the 1,000 most popular items over a period of a month. That could have led to selecting mostly items whose popularity spiked around given times during the month before going down again, while the spotlight was turning towards other items.

Such a hypothesis could explain why our implementation of Jodie is reaching up to 60% higher MRR on LastFM compared with the original implementation. The main difference between our implementation and the

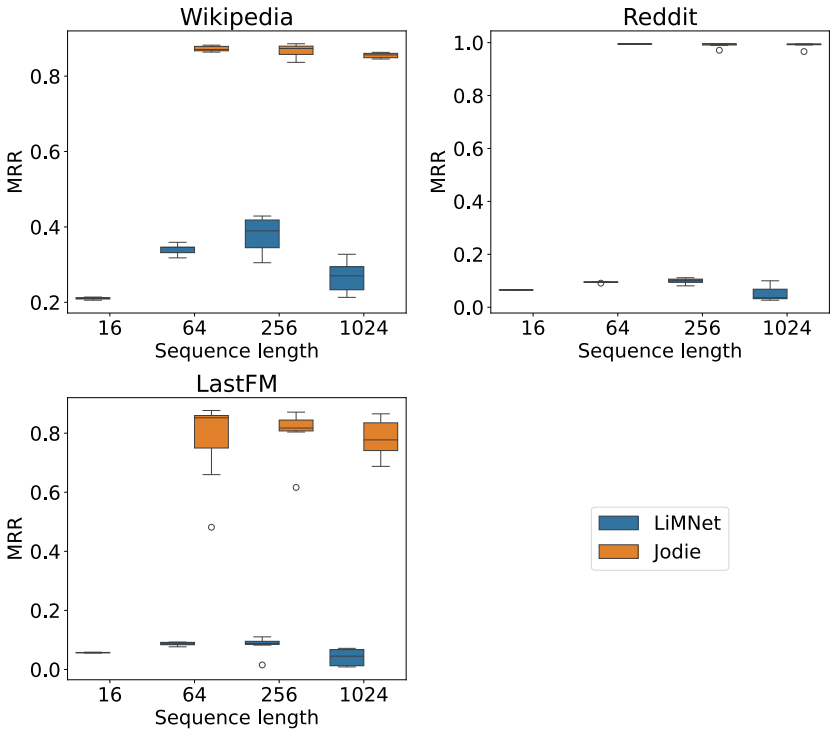

Figure 4.5: Effect of the sequence length on the models' performances.

one presented in the paper is the batching algorithm; in this project, interactions are processed in chronological order, while in [9] they are processed following the t-batch algorithm. The t-batch algorithm groups interactions with common items or users together, leading to a more localized information sharing mechanism.

Chapter 5
Conclusions

The goal of this thesis was to understand the potential of the LiMNet model for interaction prediction task on a user-item dynamical network. This research question led to the creation of an evaluation framework for this task, the evaluation of variants of the LiMNet model on this framework, along with re-implementing the Jodie model as a baseline. There are three key findings of this thesis: Firstly, LiMNet significantly underperform on the interaction prediction task compared with Jodie. Secondly, normalizing the embeddings throughout the cross-RNN mechanism improves the performance of the model. Lastly, the length of the interaction sequence processed by the models have little impact on the models results, hinting at short term global trends effect dominance over long term local preference behaviors.

5.1. Limitations

To keep the project's scope to a reasonable size, some questions had to be set aside during the research project, thus, the findings only concerns the task of user-item interaction prediction and doesn't address more general link-prediction tasks. In addition, this work focused on embedding creation with little consideration for how to most efficiently employ these embeddings. Another aspect that was left over during the research process was the performance tests, and more specifically the performance at inference. That aspect is one of the most benefit of the LiMNet architecture as demonstrated in [4], it is therefore good to keep in mind this limitation of the present project when assessing the potential of LiMNet .

5.2. Future Works

The experiment on sequence length discussed in Section 4.2 exposed a surprising and novel view on the underlying mechanisms that steer the behaviors recorded in the datasets used. Therefore, it would be very beneficial to try the models tested in this study on more suited datasets that do exhibits more complex long-term and local behaviors.

The other interesting gap we suggest to explore is the difference between the performances of the LiMNet model and the Jodie one, since both models share the same core, it is surprising to witness such a big difference in their capabilities. Exploring the design space that separates this models through an ablation study of the components of Jodie could bring more insight on which architectural decision generates such a performance leap.

5.3. Ethics and Sustainability

The progress of Machine Learning applications, which allows for leveraging big data sources at the expense of large infrastructure costs, increases the risk of inequalities by increasing the power given to the biggest institutions. However, for this project, that risk is mitigated thanks to three aspects of the LiMNet architecture. Firstly, the big selling points of this architecture are its scalability and lightweight aspect, both elements that contribute to reduce the entry cost to run such a model. Secondly, there is ongoing research to adapt this architecture into a decentralized and collaborative variant [12]. That variant may allow communities to leverage the model using distributed resources. Lastly, this project is public and thus easily accessible for legislators and law enforcers. There is an ever-increasing need to push the legislation on the use of new technologies, and research allowing legislators to take informed decisions about these subjects that still contain a lot of unknowns is valuable.

The reduced cost of computing power is also subject to potential environmental impacts. It is, however, still unclear if increasing energy efficiency leads to an actual energy saving in the long run [13].

References

[1] M. G. Rodriguez, K. Gummadi, and B. Schoelkopf, "Quantifying Information Overload in Social Media and Its Impact on Social Contagions," *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 8, no. 1, pp. 170–179, May 2014, doi: 10.1609/icwsm.v8i1.14549.

[2] J. Batista and P. Marques, "An Overview on Information and Communication Overload," *Information and Communication Overload in the Digital Age*. pp. 1–20, 2017. doi: 10.4018/978-1-5225-2061-0.ch001.

[3] L. Shahrzadi, A. Mansouri, M. Alavi, and A. Shabani, "Causes, Consequences, and Strategies to Deal with Information Overload: A Scoping Review," *International Journal of Information Management Data Insights*, vol. 4, no. 2, p. 100261, Nov. 2024, doi: 10.1016/j.jjimei.2024.100261.

[4] L. Giaretta, A. Lekssays, B. Carminati, E. Ferrari, and S. Girdzijauskas, "LiMNet : Early-Stage Detection of IoT Botnets with Lightweight Memory Networks," in *Computer Security – ESORICS 2021 : 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part I*, in Lecture Notes in Computer Science. Springer Nature, 2021. doi: 10.1007/978-3-030-88418-5_29.

[5] Z. Cui, "Classification of Financial Transactions using Lightweight Memory Networks," 2022. [Online]. Available: https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-321923

[6] R. N. Lichtenwalter, J. T. Lussier, and N. V. Chawla, "New perspectives and methods in link prediction," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, in KDD '10. Washington, DC, USA: Association for Computing Machinery, 2010, pp. 243–252. doi: 10.1145/1835804.1835837.

[7] Y. Zheng, L. Yi, and Z. Wei, "A survey of dynamic graph neural networks," *Frontiers of Computer Science*, vol. 19, no. 6, p. 196323, Dec. 2024, doi: 10.1007/s11704-024-3853-2.

[8] H. Dai, Y. Wang, R. Trivedi, and L. Song, "Deep Coevolutionary Network: Embedding User and Item Features for Recommendation," no. arXiv:1609.03675, arXiv, Feb. 2017. doi: 10.48550/arXiv.1609.03675.

[9] S. Kumar, X. Zhang, and J. Leskovec, "Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, in KDD '19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 1269–1278. doi: 10.1145/3292500.3330895.

[10] Z. Kefato, S. Girdzijauskas, N. Sheikh, and A. Montresor, "Dynamic Embeddings for Interaction Prediction," in *Proceedings of the Web Conference 2021*, in WWW '21. Ljubljana, Slovenia: Association for Computing Machinery, 2021, pp. 1609–1618. doi: 10.1145/3442381.3450020.

[11] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," 2014. [Online]. Available: https://arxiv.org/abs/1412.3555

[12] L. Giaretta, A. Lekssays, B. Carminati, E. Ferrari, and S. Girdzijauskas, "Metasoma: Decentralized and CollaborativeEarly-Stage Detection of IoT Botnets."

[13] K. Gillingham, D. Rapson, and G. Wagner, "The Rebound Effect and Energy Efficiency Policy," *Review of Environmental Economics and Policy*, vol. 10, no. 1, pp. 68–88, 2016, doi: 10.1093/reep/rev017.

TRITA – EECS-EX 2024:0000
Stockholm, Sweden 2025
www.kth.se