الجمهورية الشعبية الديمقراطية الجزائرية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement supérieur et de la Recherche scientifique

---

Ecole Nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

TELECOM SudParis

IP PARIS

Département Génie Industriel

End of Study Project Dissertation

for **Obtaining State Engineer's Degree**
in **Industrial Engineering**

Option : **Data Science & Artificial Intelligence**

Optimization of Recommender Systems using Knowledge Graphs

Presented by:

MOKEDDEM Boualem GUIA Hocine Islam

Defended on **29 Juin, 2025**, before a jury composed of :

Mr. Iskander ZOUAGHI	ENP - President
Mr. Oussama ARKI	ENP - Examiner
Ms. Samia BELDJOUDI	ENP - Supervisor
Ms. Amel BOUZEGHOUB	IPP - Supervisor
Mr. Vincent GAUTHIER	IPP - Supervisor

ENP 2025

الجمهورية الشعبية الديمقراطية الجزائرية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement supérieur et de la Recherche scientifique

---

Ecole Nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

TELECOM SudParis

IP PARIS

Département Génie Industriel

End of Study Project Dissertation

for **Obtaining State Engineer's Degree**
in **Industrial Engineering**

Option : **Data Science & Artificial Intelligence**

Optimization of Recommender Systems using Knowledge Graphs

Presented by:

MOKEDDEM Boualem GUIA Hocine Islam

Defended on **29 Juin, 2025**, before a jury composed of :

Mr. Iskander ZOUAGHI	ENP - President
Mr. Oussama ARKI	ENP - Examiner
Ms. Samia BELDJOUDI	ENP - Supervisor
Ms. Amel BOUZEGHOUB	IPP - Supervisor
Mr. Vincent GAUTHIER	IPP - Supervisor

ENP 2025

الجمهورية الشعبية الديمقراطية الجزائرية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement supérieur et de la Recherche scientifique

---

Ecole Nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

TELECOM SudParis
IP PARIS

Département Génie Industriel

Mémoire de Projet de Fin d'Études

pour **l'obtention du diplôme d'ingénieur d'État**
en **Génie Industriel**

Option : **Science des données & Intelligence artificielle**

Optimisation des systèmes de recommandation utilisant des graphes de connaissances

---

Présenté par :

MOKEDDEM Boualem & GUIA Hocine Islam

Soutenu le **29 Juin 2025**, devant un jury composé de :

Mr. Iskander ZOUAGHI	ENP - Président
Mr. Oussama ARKI	ENP - Examinateur
Ms. Samia BELDJOUDI	ENP - Encadrante
Ms. Amel BOUZEGHOUB	IPP - Encadrante
Mr. Vincent GAUTHIER	IPP - Encadrant

ENP 2025

الملخص

تُعاني أنظمة التوصيات المكانية والزمانية من صعوبة استغلال السياق الجغرافي على أكمل وجه. ومع ذلك، لا تزال المعلومات المتعلقة بالمناطق الإدارية والعلاقات بين النقاط المهمة غير مستغلة بشكل كافٍ إلى حد كبير. تقترح هذه الأطروحة دمج رسم بياني للمعرفة الحضرية لسد هذه الفجوة.

تمت مقارنة العديد من نماذج التنبؤ، مثل: تدريب كل بنية معمارية مرتين، دون أي معلومات بيانية، وعن طريق حقن تضمينات المعرفة كسمات ثابتة أو حالات أولية.

من خلال الجمع بين التفكير الرمزي (رسم بياني للمعرفة) والتعلم العميق المكاني الزماني، تُظهر هذه الدراسة أن السياق الجغرافي الواضح يُحسّن دقة توصيات الأماكن بشكل كبير.

الكلمات المفتاحية --- نظام التوصية، الرسم البياني للمعرفة، التضمين.

Résumé

Les systèmes de recommandation spatiales et temporelles peinent à exploiter pleinement le contexte géographique. Pourtant, l'information sur les zones administratives et les relations entre points d'intérêt reste largement sous-utilisée. Cette thèse propose d'intégrer un graphe de connaissances urbain pour combler cette lacune. Plusieurs modèles de prédiction sont comparés, telle que Chaque architecture est entraînée deux fois, sans aucune information de graphe, et en injectant les embeddings KG comme caractéristiques statiques ou comme états initiaux. En combinant raisonnement symbolique (graphe de connaissances) et apprentissage profond spatio-temporel, cette étude montre qu'un contexte géographique explicite enrichit significativement la précision des recommandations de lieux.

Mots-clés— Système de recommandation, Graphe de connaissance, Embedding.

Abstract

Spatial and temporal recommendation systems struggle to fully exploit the geographic context. However, information on administrative areas and relationships between points of interest remains largely underutilized. This thesis proposes to integrate an urban knowledge graph to fill this gap. Several prediction models are compared, such as: Each architecture is trained twice, without any graph information, and by injecting KG embeddings as static features or initial states. By combining symbolic reasoning (knowledge graph) and spatiotemporal deep learning, this study shows that an explicit geographic context significantly improves the accuracy of place recommendations.

Keywords— Recommendation System, Knowledge graph, Embedding

Acknowledgement

In the name of Allah, the Most Gracious, the Most Merciful. All praise is due to Allah, Lord of the worlds, for granting us the strength, knowledge, and opportunity to undertake this research study and to persevere and complete it satisfactorily. Without His blessings, this achievement would not have been possible.

We would like to express our profound gratitude to our research supervisors, Professors Mme. Amel Bouzeghoub and Mme. Samia Beldjoudi, for their exceptional guidance, unwavering support, and profound expertise throughout our research journey. Their insightful feedback pushed us to sharpen our thinking and brought our work to a higher level. Their patience, motivation, and immense knowledge were instrumental in helping us navigate the challenges of academic research and writing.

Lastly, we wish to express our sincere gratitude to all those who have contributed to the successful completion of this thesis. Whether through academic support, technical assistance, or words of encouragement, your contributions have not gone unnoticed. This accomplishment is as much a result of your collective support as it is of our hard work.

Contents

List of Figures . . . . . .
List of Tables . . . . . .
List of Acronyms . . . . . .

General Introduction **11**

1 Machine–Learning Foundations for Recommender Systems **13**
Introduction 13
1.1 From Data to Models: What "Learning" Means 13
1.1.1 What is a Learning Task? 13
1.1.2 Kinds of Supervision Signals 14
1.1.3 From Datasets to Generalisation 14
1.1.4 Optimisation in Practice 15
1.2 Supervised and Unsupervised Paradigms 15
1.2.1 Supervised Learning: Predicting Known Targets 15
1.2.2 Unsupervised Learning: Discovering Latent Structure 16
1.2.3 When to Choose Which Paradigm? 16
1.2.4 Bridging the Gap: Semi- and Self-Supervised Approaches 16
1.3 Deep–Learning Building Blocks 18
1.3.1 Fully Connected (Dense) Networks 18
1.3.2 Convolutional Layers 18
1.3.3 Recurrence: Sequence Modelling with RNNs & LSTMs 18
1.3.4 Attention Mechanisms 19
1.3.5 Over-Parameterisation and Regularisation 19
1.4 Spatio–Temporal Signals in User Behaviour 19
1.4.1 Spatial Locality 19
1.4.2 Temporal Periodicity 21
1.4.3 Cross-Effects: Space $\times$ Time Coupling 21
1.4.4 Typical Data Granularity 21
1.4.5 Challenges Arising from Spatio–Temporal Data 21
1.5 Model-Evaluation Methodology 22
1.5.1 Task-Aligned Metrics 22
1.5.2 Choosing a Data-Splitting Strategy 23
1.5.3 Statistical Significance & Uncertainty 23
1.6 Challenges & Open Questions 24
1.6.1 Data Sparsity & Cold-Start Dilemmas 24

1.6.2 Temporal Covariate Shift & Concept Drift . . . 25
1.6.3 Scalability Constraints . . . 25
1.6.4 Privacy & Security . . . 25
1.6.5 Fairness & Bias . . . 25
1.6.6 Interpretability & User Trust . . . 26
Conclusion . . . 26

2 Graph–Centric Representation Learning **27**
Introduction . . . 27
2.1 Essentials of Graph Theory . . . 27
2.1.1 Graphs, Nodes and Edges . . . 27
2.1.2 Beyond Simple Graphs: Richer Edge Semantics . . . 28
2.1.3 Walks, Paths, Trails and Cycles . . . 28
2.1.4 Connectivity Notions . . . 28
2.1.5 Communities and Modular Structure . . . 29
2.2 From Graphs to Vectors: Embedding Methods . . . 29
2.2.1 Goals of Embeddings . . . 29
2.2.2 High-Level Families of Embedding Algorithms . . . 30
2.2.3 Training Objectives . . . 31
2.3 Graph Machine Learning . . . 31
2.3.1 Message-Passing Neural Networks (GNNs) and Variants . . . 32
2.3.2 Static vs. Dynamic Graphs; Temporal Message Passing . . . 34
2.3.3 Expressiveness, Over-Smoothing, and Scalability . . . 35
2.4 Graph-Enhanced Recommendation: A Conceptual View . . . 37
2.4.1 Capturing Multi-Hop Relational Context . . . 37
2.4.2 Handling Heterogeneity Through Typed Edges and Nodes . . . 38
2.4.3 Advantages over Purely Sequence- or Matrix-Based Approaches . . . 40
2.5 Related Work . . . 42
2.6 Evaluation & Benchmarking in Graph ML . . . 43
2.6.1 Link Prediction and Node Classification . . . 43
2.6.2 Temporal Link Forecasting . . . 45
2.6.3 Influence of Graph Sparsity and Dynamic Updates on Metrics . . . 47
Conclusion . . . 48

3 Spatio-temporal Recommendation systems optimization using knowledge graphs **49**
Introduction . . . 49
3.1 Geographical Overview and Administrative Decomposition of Tokyo and San Francisco . . . 50
3.1.1 Tokyo, Japan . . . 50
3.1.2 San Francisco, California, USA . . . 51
3.2 Database Construction for Urban Knowledge Graphs . . . 52
3.2.1 Spatial Database Development . . . 52
3.2.2 Hierarchical Structure Design . . . 52
3.2.3 Handling Tokyo's Neighbourhood Challenges . . . 52

3.2.4 Mobility Database Integration . . . 53
3.2.5 Data-Quality Assurance . . . 54
3.3 Knowledge Graph Construction Process . . . 54
3.3.1 Data Preparation . . . 54
3.3.2 Spatial Relationship Extraction . . . 55
3.3.3 Knowledge Graph Assembly . . . 56
3.3.4 Final Output . . . 56
3.4 Knowledge Graph Embedding Generation Process . . . 56
3.4.1 Data Preparation . . . 56
3.4.2 Model Configuration . . . 57
3.4.3 Training Procedure . . . 57
3.4.4 Output Embeddings . . . 57
3.4.5 Results of Knowledge-Graph Embedding Generation . . . 57
3.5 LSTM-Based Next-POI Predictor: Data Pipeline and Network Layout . . . 60
3.5.1 Temporal Batches (t-batches) . . . 60
Run-Time Hyper-Parameters . . . 61
3.5.2 Pipeline Overview . . . 61
3.6 TGAT Model Architecture . . . 62
3.6.1 Core Components . . . 62
3.6.2 Training Protocol . . . 63
3.6.3 Evaluation Metrics . . . 63
3.6.4 Dataset Format . . . 63
3.6.5 Default Configuration . . . 63
3.7 TGN-Attention (TGN-attn) Model Architecture . . . 64
3.7.1 Core Components . . . 64
3.7.2 Training Pipeline . . . 65
3.7.3 Evaluation Protocol . . . 65
3.7.4 Command-Line Invocation . . . 66
3.8 Results and findings . . . 66
Conclusion . . . 68

General Conclusion **69**

Bibliography **72**

Annex **73**

List of Figures

3.1 Project's pipeline 50
3.2 The images above represent the map of Japan and Tokyo indicating different sub-areas 51
3.3 The images above represent the map of California and San Francisco 51
3.4 Urban Knowledge Graph construction 55
3.5 2-D t-SNE projections of 32-D POI embeddings, coloured by district. TransE consistently forms tighter, more separable clusters than GIE, especially in Tokyo, which aligns with its lower MR and higher Hits@k. 59
3.6 TGAT model architecture 62
3.7 Flow of operations of TGN used to train the memory-related modules. Raw Message Store stores the necessary raw information to compute messages, i.e. the input to the message functions, which we call raw messages, for interactions which have been processed by the model in the past. This allows the model to delay the memory update brought by an interaction to later batches. At first, the memory is updated using messages computed from raw messages stored in previous batches (1, 2, 3). The embeddings can then be computed using the just updated memory (grey link) (4). By doing this, the computation of the memory-related modules directly influences the loss (5, 6), and they receive a gradient. Finally, the raw messages for this batch interactions are stored in the raw message store (6) to be used in future batches. 64
3.8 JODIE model architecture 74

List of Tables

1.1 Supervision signals and their typical treatment 14
1.2 Canonical supervised tasks, losses, and examples 15
1.3 Unsupervised tasks, objectives, and applications 16
1.4 Selecting a learning paradigm based on practical context 17
1.5 Regularisation strategies for deep networks 20
1.6 Coupled space–time phenomena and modelling responses 21
1.7 Granularity of spatio-temporal data and associated noise 22
1.8 Task-aligned metrics for classifier and recommender evaluation 23
1.9 Facets of sparsity and cold-start in mobility recommendation 24
1.10 Scalability pain points in spatio-temporal recommendation 25
1.11 Fairness hazards in geospatial recommendation 26

2.1 Richer graph flavours beyond the unweighted, static case 28
2.2 Traversal notions and their uses 28
2.3 Popular community-detection algorithms 29
2.4 Previous models for deep learning on continuous-time dynamic graphs are specific case of our TGN framework. Shown are multiple variants of TGN used in our ablation studies. method (l,n) refers to graph convolution using l layers and n neighbors. yuses t-batches. $*$ uses uniform sampling of neighbors, while the default is sampling the most recent neighbors. zmessage aggregation not explained in the paper. k uses a summary of the destination node neighborhood (obtained through graph attention) as additional input to the message function. 42

3.1 Relational schema for both cities 53
3.2 Final mobility data after enrichment 53
3.3 Basic statistics of the Tokyo and San Francisco UrbanKG datasets 54
3.4 TransE hyper-parameter choices 57
3.5 Baseline network layout 61
3.6 TGN-attn building blocks 65
3.7 Training configuration 65
3.8 Full models comparison across cities. “T” and “I” denote transductive and inductive variants. 66

List of Acronyms

AUC: Area Under the Curve
BCE: Binary Cross-Entropy
CNN: Convolutional Neural Network
DL: Deep Learning
GAT: Graph Attention Network
GCN: Graph Convolutional Network
GIE: Geometry Interaction Embeddings
GNN: Graph Neural Network
GRU: Gated Recurrent Unit
GPU: Graphics Processing Unit
JODIE: JOint Dynamic userItem Embedding
KG: Knowledge Graph
LSTM: Long Short-Term Memory
MSE: Mean Squared Error
MR: Mean Rank
MRR: Mean Reciprocal Rank
MPNNs: Message Passing Neural Networks
NDCG: Normalized Discounted Cumulative Gain
OHE: One-Hot Encoding
POI: Point of Interest
RNN: Recurrent Neural Network
SF: San Francisco
TGAT: Temporal Graph Attention Network
TGN: Temporal Graph Network
t-SNE: t-Distributed Stochastic Neighbor Embedding
UUKG: Unified Urban Knowledge Graph

General Introduction

Context

The past decade has seen an explosion of location-aware digital services—ride-hailing, food delivery, urban exploration apps—that must predict not only what a user will do, but where and when they will go next. Conventional recommender systems excel at modeling user–item affinities in static domains such as movies or products, yet human mobility unfolds on a richly structured, evolving map of points of interest, administrative districts, and semantic landmarks. Pure sequence or matrix-based methods struggle to capture the hierarchical geography of a city—its wards and neighborhoods—or the multi-relational dependencies between restaurants, transit hubs, and popular attractions.

Knowledge graphs offer a natural remedy, linking entities (POIs, areas, categories) via explicit relations (*located_in*, *nearby*, *has_category*) and enabling symbolic reasoning over urban structure. At the same time, advances in graph embedding and dynamic graph neural networks promise to fuse these symbolic structures with deep, time-aware representations. By projecting a city's spatial hierarchy into a continuous vector space and feeding those vectors into spatio-temporal predictors, we can hope to deliver recommendations that respect both the shape of the city and the rhythms of human movement—yielding more accurate, context-sensitive predictions, even for new users or rarely visited locales.

Problem Statement

This thesis investigates whether embedding an explicit urban knowledge graph into different prediction architectures can measurably improve next-POI recommendations. We ask:

1. How can we build a reproducible, hierarchical knowledge graph that captures districts, neighborhoods, POIs, and their semantic relations for major cities?

2. Which graph-embedding techniques best distill that structure into usable vectors?

3. Can these vectors, when injected as static features or initialization states, boost the performance of spatio-temporal models—from sequence learners (LSTM) to attention-based (TGAT) and memory-augmented networks (TGN-attn)—without compromising training stability or generalization?

Thesis Outline

This engineering thesis is structured into three main chapters:

- **Chapter 1:** Machine–Learning Foundations for Recommender Systems
  Introduces core ML concepts—supervised vs. unsupervised learning, deep-learning blocks, and evaluation metrics—focused on modeling "where" and "when."

- **Chapter 2:** Graph–Centric Representation Learning
  Exploration of graph-based methods as a growing trend in recommendation systems.

- **Chapter 3:** Spatio-temporal Recommendation systems optimization using knowledge graphs
  Details the end-to-end pipeline—from building the UrbanKG for Tokyo/SF to training embeddings and integrating them into LSTM, TGAT, and TGN-attn models—evaluating the impact of KG signals.

Chapter 1

Machine–Learning Foundations for Recommender Systems

Introduction

Recommending the next place a user will visit requires more than matching past clicks—it demands a solid grasp of how machines learn from data and how to measure their performance. In this chapter, we introduce the core concepts of machine learning—from the distinction between supervised, unsupervised, and reinforcement signals to the building blocks of deep networks—and show how "where" and "when" can be treated as first-class inputs. We also establish rigorous evaluation protocols, so that every model comparison is fair, reproducible, and statistically sound.

1.1 From Data to Models: What "Learning" Means

1.1.1 What is a Learning Task?

At its heart, a learning task is the search for a function

$$f_{\boldsymbol{\theta}} \,:\, \mathcal{X} \longrightarrow \mathcal{Y}, \tag{1.1}$$

parameterised by $\boldsymbol{\theta}$, that maps each input $x \in \mathcal{X}$ to an output $y \in \mathcal{Y}$. The pair $(\mathcal{X}, \mathcal{Y})$ appears in many guises:

- **Classification** — $\mathcal{Y}$ is a finite set of discrete labels (e.g., *spam/not-spam*, *image categories*, *user-preferred POI*).
- **Regression** — $\mathcal{Y} \subseteq \mathbb{R}^d$ is continuous (e.g., *house price*, *journey time*).
- **Ranking / Recommendation** — the model produces an ordering over candidate items; mathematically this is often expressed through a real-valued score that induces the ranking.

Learning proceeds by *empirical risk minimisation*: we select $f_{\boldsymbol{\theta}}$ from a hypothesis space (all functions representable by the chosen architecture) so that a loss $\mathcal{L}\Big(f_{\boldsymbol{\theta}}(x), y\Big)$ is small *on average* under the data-generating distribution. In practice, the expectation is approximated by a finite sample—the dataset.[1]

1.1.2 Kinds of Supervision Signals

Choosing the correct learning signal dictates the loss, optimisation strategy, and evaluation protocol. Table 1.1 summarises the spectrum.

Signal type	**What the learner sees**	**Illustrative examples**	**Typical algorithms**
Fully labelled	Explicit (x, y) pairs	Image with its class; trajectory with ground-truth next location	Logistic regression, deep CNNs, random forests
Partially / weakly labelled	Noisy, incomplete, or coarse labels	Web-scale image tags, distant supervision in NLP	Semi-supervised learning, label propagation
Implicit feedback	Behavioural traces interpreted as preference strength	Click-through, dwell time, check-ins	Bayesian personalised ranking, implicit MF
Reward-driven	Scalar reward after each action	Online ad bidding, game playing	Q-learning, policy gradients
Unlabelled	Inputs only	Raw text corpora, sensor streams	Contrastive learning, autoencoders

Table 1.1: Supervision signals and their typical treatment

1.1.3 From Datasets to Generalisation

Robust practice enforces a strict separation:

Training set optimises parameters $\boldsymbol{\theta}$;

Validation set guides hyper-parameter tuning and early stopping;

Test set provides an *unbiased* estimate of generalisation.

For small datasets or hyper-parameter sensitive models, k-fold cross-validation averages performance over multiple folds. In temporal domains, a chronological hold-out (train on earlier times; validate/test on later times) prevents information leakage and mirrors deployment.

Two antagonistic errors loom large:

- **Underfitting** — hypothesis space too restricted ⇒ high bias.
- **Overfitting** — model memorises idiosyncrasies ⇒ high variance.

Classical counter-measures include ℓ_2 weight decay, dropout, data augmentation, and ensembling.[14]

1.1.4 Optimisation in Practice

Most modern models are trained by minimising empirical loss via stochastic gradient descent (SGD) or adaptive variants (Adam, RMSProp). Mini-batch updates yield noisy yet inexpensive gradients, trading variance for speed. Although objectives are typically non-convex—especially in deep networks—SGD empirically finds flat, low-loss basins that generalise well.

1.2 Supervised and Unsupervised Paradigms

Modern machine–learning problems fall into two broad families—*supervised* and *unsupervised*—depending on whether ground-truth outputs are available during training. This section explains when each paradigm is appropriate and how its principal task types are formalised and evaluated.[11]

1.2.1 Supervised Learning: Predicting Known Targets

Table 1.2 resumes the principal supervised tasks in ML :

Task type	$\mathcal{Y}$ **(output space)**	$\mathcal{L}$ **(typical loss)**	**Illustrative examples**
Regression	$\mathbb{R}$ or $\mathbb{R}^d$ (continuous)	MSE, MAE	Predicting travel time, house price, user rating
Classification	Finite label set $\{1, \ldots, K\}$	Cross-entropy, hinge	Email spam detection, image category, churn
Ranking / recommendation	Permutation of items or scores inducing an order	Pairwise hinge, BPR, listwise cross-entropy	Top-N POI recommendation, search-result ordering

Table 1.2: Canonical supervised tasks, losses, and examples

Why use supervised learning?

- An explicit target must be predicted.

- Performance can be quantified unambiguously (e.g., RMSE, accuracy, NDCG, MRR).
- Trained models can be deployed directly for prediction or decision-making.

1.2.2 Unsupervised Learning: Discovering Latent Structure

Table 1.3 resumes the principal unsupervised tasks in ML :

Task type	**Objective**	**Canonical algorithms**	**Use cases**
Clustering	Partition data into homogeneous groups	k-means, DBSCAN, spectral, hierarchical	Market segmentation, trajectory pattern mining
Density estimation / generative modelling	Approximate $p(x)$	Gaussian mixtures, normalising flows, VAEs	Anomaly detection, synthetic data generation
Representation learning	Learn compact, informative features $z = g_\phi(x)$	Autoencoders, contrastive (SimCLR, MoCo), word2vec	Downstream supervised tasks, visualisation, transfer

Table 1.3: Unsupervised tasks, objectives, and applications

Why use unsupervised learning?

- Labels are unavailable, expensive, or the goal is exploratory.
- Reveals hidden structure that may seed later supervised models (pre-training) or support decision-making when explicit targets are unknown.

1.2.3 When to Choose Which Paradigm?

Table 1.4 explains, through examples, how to choose between different paradigms, previously explained :

1.2.4 Bridging the Gap: Semi- and Self-Supervised Approaches

- **Semi-supervised learning** mixes a small labelled subset with a large unlabelled pool (pseudo-labelling, consistency regularisation).

Scenario	**Most suitable paradigm**	**Rationale**
Millions of user–item clicks; goal is to rank items	Supervised ranking	Abundant implicit labels; ranking metrics define success.
GPS traces with no stated intent; aim is to group similar movement patterns	Unsupervised clustering	No ground truth; objective is to expose latent mobility motifs.
Few labelled sequences; want a deep model	Unsupervised representation learning → supervised fine-tuning	Self-supervised pre-training mitigates label scarcity and boosts generalisation.

Table 1.4: Selecting a learning paradigm based on practical context

- **Self-supervised learning** constructs proxy tasks (masking, contrastive pairs) whose "labels'' are derived from the data itself, enabling the model to learn rich features that transfer to fully supervised tasks.

1.3 Deep–Learning Building Blocks

Deep learning can be viewed as a toolkit of *composable, differentiable layers* stacked and trained end-to-end. Mastering the vocabulary of those layers is essential before we turn to specialized architectures in later chapters.[8]

1.3.1 Fully Connected (Dense) Networks

Definition. A dense layer applies an affine transformation followed by a non–linear activation:

$$\mathbf{h} \;=\; \sigma\Big(\mathbf{W}\mathbf{x} + \mathbf{b}\Big), \tag{1.2}$$

where σ is typically ReLU, GELU, or `tanh`.

Role. Captures *global* interactions between all input dimensions and constitutes the basic building block of multilayer perceptrons (MLPs).

Strengths / weaknesses. Highly expressive yet parameter–intensive and *position-agnostic*; often supplanted by convolution or attention when local structure matters.

1.3.2 Convolutional Layers

- **Local receptive field.** A kernel $\mathbf{K} \in \mathbb{R}^{k \times k \times C_{\text{in}} \times C_{\text{out}}}$ slides over the input, re-using weights to detect spatially local patterns.
- **Translation equivariance & parameter sharing.** The same weights process every spatial location, reducing overfitting risk and computation—crucial for images, audio, and grid-like sensor data.
- **Variants.** 1-D and 3-D convolutions, dilated and depthwise-separable kernels, residual connections (ResNet) that stabilise very deep stacks.

1.3.3 Recurrence: Sequence Modelling with RNNs & LSTMs

Idea. Maintain a hidden state $\mathbf{h}_t$ that integrates all previous timesteps:

$$\mathbf{h}_t \;=\; g\Big(\mathbf{x}_t,\, \mathbf{h}_{t-1};\, \boldsymbol{\theta}\Big). \tag{1.3}$$

Long Short-Term Memory (LSTM). Adds input, forget, and output gates plus a cell state $\mathbf{c}_t$ to mitigate vanishing gradients and capture long-range dependencies.

Enhancements. Bidirectional and stacked RNNs improve context capture; however, sequential computation hinders parallelism and struggles with very long sequences without architectural tweaks (attention, memory).

1.3.4 Attention Mechanisms

Core principle. Given a query $\mathbf{q}$, keys K and values V,

$$\text{Attention}(\mathbf{q}, K, V) \;=\; \text{softmax}\left(\frac{\mathbf{q}K^\top}{\sqrt{d}}\right)V. \tag{1.4}$$

- **Self-attention & the Transformer.** Every position in a sequence attends to every other, enabling global dependency modelling with full parallelism.
- **Multi-head attention.** Parallel projections capture diverse relational subspaces; heads are concatenated and linearly mixed.
- **Extensions.** Cross-modal attention (vision–language), sparse and efficient variants for very long sequences.

1.3.5 Over-Parameterisation and Regularisation

Different regularization technics of deep networks are explained in Table 1.5.[19]

1.4 Spatio–Temporal Signals in User Behaviour

Recommender systems that serve *physical experiences*—next-place suggestion, mobility assistance, local search, contextual advertising—must reason about *where* users are and *when* they act. This section surveys the empirical regularities and data characteristics that make spatio-temporal information indispensable.[7]

1.4.1 Spatial Locality

First law of geography. "Everything is related to everything else, but near things are more related than distant things''. In practice, the probability that a user visits a point of interest (POI) decays with distance—often captured by a power-law or exponential kernel.

- **Neighbourhood effects.** Home and work anchors act as spatial hubs; POIs within a few kilometres dominate weekday activity.
- **Semantic proximity.** Restaurants cluster near shopping malls; trajectories follow micro-regions rather than arbitrary map tiles.

Design implication: models must embed geodesic distance or region adjacency so that "nearby'' candidates receive higher prior scores.

Concept / Technique	Motivation	Practical notes
Over-parameterisation	Very deep or wide networks often generalise better despite having more parameters than datapoints.	GPT-style Transformers; large vision models.
Dropout	Randomly zeroes activations, forcing robustness and reducing co-adaptation.	Typical rates 0.1–0.5; applicable to dense, conv., attention layers.
Batch Normalisation	Normalises layer inputs to zero mean / unit variance, accelerating convergence and acting as regulariser.	Widely used in CNNs; LayerNorm preferred in Transformers.
Weight decay	Adds an ℓ_2 penalty $\lambda\|\boldsymbol{\theta}\|_2^2$ to discourage large weights.	Built into optimisers like AdamW.
Early stopping & data augmentation	Halt training when validation loss rises; augment data to improve invariance.	Standard in computer vision and NLP pre-training.

Table 1.5: Regularisation strategies for deep networks

1.4.2 Temporal Periodicity

- **Diurnal cycles.** Commute peaks (07–09 h, 17–19 h), lunchtime spikes, late-evening leisure.
- **Weekly patterns.** Weekdays versus weekends, Friday-night entertainment, Sunday shopping bans (in some locales).
- **Seasonality & events.** Holiday migrations, weather-induced shifts, one-off festivals modulate baseline patterns.

Design implication: predictive models benefit from explicit time encodings (hour-of-day, day-of-week, Fourier bases) or learned temporal embeddings.

1.4.3 Cross-Effects: Space × Time Coupling

Spatial and temporal signals are *not* independent (see Table 1.6)

Observation	**Example**	**Modelling strategy**
Temporal locality shrinks space	Lunchtime (12–14 h): users rarely travel >1 km from work.	Time-conditioned distance kernels; context-aware candidate pruning.
Event-driven reconfiguration	Stadium concerts centre visit density on venue & transport hubs.	Dynamic graphs or temporal attention to re-weight edges in real time.

Table 1.6: Coupled space–time phenomena and modelling responses

1.4.4 Typical Data Granularity

Granularity drives both model choice (grid-based convolution vs. graph message passing) and privacy risk (coarser regions mitigate re-identification). Table 1.7 details the granularity of spatiotemporal data :

1.4.5 Challenges Arising from Spatio–Temporal Data

- **Irregular sampling** — GPS fixes arrive at uneven intervals; users may disable location services.
- **Sparsity & cold-start** — New POIs or travellers have few observations.
- **Multi-scale behaviour** — Long-distance travel co-exists with micro-mobility inside venues.
- **Privacy constraints** — Regulations (GDPR, CCPA) restrict fine-grained location retention and mandate anonymisation.[18]

Dimension	Min. unit	Common representations	Noise sources
Spatial			
Point	GPS fix (5–20 m); cell-tower centroid (100–300 m)	WGS-84 coords; S2/H3 cells; road-network node IDs	Multipath, urban canyons, device heterogeneity
Region	Administrative polygon, census tract, POI category	Shapefiles, GeoJSON, OSM IDs	Boundary updates, inconsistent zoning
Temporal			
Time-stamp	Unix epoch ms	ISO-8601 strings; integer epoch	Clock drift, time-zone misalignment
Interval / session	Start	end times	Aggregated logs; stay-point detection
Sessionisation-heuristic variance			

Table 1.7: Granularity of spatio-temporal data and associated noise

1.5 Model-Evaluation Methodology

Rigorous evaluation is the linchpin of empirical machine learning: without a principled protocol, we cannot tell improvement from noise. This section organises the *what* (metrics), *how* (experimental design), and *how sure* (statistical testing) of model assessment so that later chapters can compare recommendation algorithms on solid ground.[5]

1.5.1 Task-Aligned Metrics

The following table 1.8 provides an overview for few Task-aligned metrics for classifier and recommender evaluation

Key distinctions.

- **Point- vs list-wise.** AUC judges pairwise ordering; NDCG inspects whole lists.
- **Top-k focus.** Mobile screens show ≤ 5 results—metrics rewarding early precision (MRR, NDCG@10) matter more than global accuracy.

Metric family	Metric	Definition (core idea)	Range / intuition	When to use
Classification	Accuracy	$\frac{1}{N}\sum_i \mathbb{1}[y_i = \hat{y}_i]$	0–1 (higher = better)	Balanced classes; simple baselines
	ROC-AUC	$\Pr\big(\text{positive rank} > \text{negative}\big)$	0.5 (random) → 1	Imbalanced binary labels
Ranking / Rec.	Hit@k	$\mathbb{1}[\text{gt} \in \text{top-}k]$	0–1	User cares if item appears anywhere in list
	MRR	$\frac{1}{N}\sum_i \frac{1}{\text{rank}_i}$	0–1; emphasises early hits	Single relevant item per query (next-POI)
	NDCG	$\frac{1}{N}\sum_i \frac{\text{DCG}_i}{\text{IDCG}_i}$, $\text{DCG} = \sum_j \frac{2^{rel_{ij}}-1}{\log_2(j+1)}$	0–1; graded relevance	Multiple graded signals (ratings, dwell-time)

Table 1.8: Task-aligned metrics for classifier and recommender evaluation

1.5.2 Choosing a Data-Splitting Strategy

a) **Random k-fold cross-validation** — maximises data use, averages variance; however it breaks temporal order and risks leakage for sequential tasks.

b) **User-level split (leave-users-out)** — removes entire users from training to test cold-start generalisation.

c) **Temporal hold-out (chronological)** — train on $[t_0, t_c)$, validate on $[t_c, t_v)$, test on $[t_v, t_f]$; mandatory for next-item or mobility prediction.

d) **Rolling-window evaluation** — re-train periodically on sliding window $[t_i,\, t_{i+w}] \rightarrow$ test at t_{i+w}, capturing model ageing and concept drift.

1.5.3 Statistical Significance & Uncertainty

Paired significance tests.

- *Paired t-test* — assumes normality; works for accuracy, AUC.
- *Wilcoxon signed-rank* — non-parametric alternative.
- *Bootstrap / permutation* — resample user-level results; robust for heavy-tailed metrics like NDCG.

Confidence intervals. For user-level mean $\hat{\mu}$, a 95% CI via studentised bootstrap is

$$\hat{\mu} \pm t_{0.975,\,N-1}\,\frac{s}{\sqrt{N}}, \tag{1.5}$$

where s is the sample standard deviation.

Multiple comparisons. Control family-wise error (Bonferroni, Holm) or false-discovery rate (Benjamini–Hochberg) when >2 models are compared.

Effect size. Report relative lift (e.g., +8% NDCG@10) or Cohen's d alongside p-values; statistical significance without practical impact is misleading.

1.6 Challenges & Open Questions

Even with a robust learning pipeline, spatio–temporal recommender systems confront obstacles that the remainder of this thesis seeks to address. Cataloguing those obstacles—technical, statistical, and societal—clarifies *why* subsequent chapters introduce knowledge graphs and dynamic-graph models, without yet revealing *how* the proposed solution works.[13]

1.6.1 Data Sparsity & Cold-Start Dilemmas

The following table 1.9 explains the effect of sparsity and cold-start in mobility recommendation

Manifestation	**Why it hurts**	**Illustrative example**	**Current mitigation ideas**
User cold-start	New or infrequent users lack interaction history, crippling personalised ranking.	Traveller installs a transit app in a new city; no past trajectory.	Profile transfer, demographic priors, zero-shot embeddings, cross-domain hints
Item cold-start	Newly opened POIs rarely appear in logs; popularity bias buries them.	Pop-up café first listed on OSM yesterday.	Content-based priors, geo-graph proximity heuristics
Long-tail sparsity	>80 % of POIs receive < 5 visits, skewing label distributions.	Rural gas stations vs. downtown malls.	Re-weighted losses, negative sampling, few-shot meta-learning

Table 1.9: Facets of sparsity and cold-start in mobility recommendation

Open question: Can relational context—"this café is inside a lively market'' or "this trailhead connects to a scenic lookout''—substitute for missing direct observations?

1.6.2 Temporal Covariate Shift & Concept Drift

Definition. The joint distribution $p_t(x, y)$ changes with clock time t.

- **Seasonal demand** — ski resorts spike in winter.
- **External shocks** — lockdown policies reshape mobility overnight.

Evaluation hazard. Models trained on 2019 logs can appear accurate offline yet fail post-2020.
Research frontier. Continual learning, online adaptation, and temporal-aware validation splits (Section 1.5.2).

1.6.3 Scalability Constraints

Dimension	Order of magnitude today	Stress point
Spatial resolution	10^7 GPS fixes / day in one metropolis	Real-time nearest-neighbour search
Graph size	10^8 entities $\times$ 10^9 edges (city-wide KG)	Memory-bound message passing
Model complexity	10^{10} parameters (large Transformers)	Distributed training, carbon footprint

Table 1.10: Scalability pain points in spatio-temporal recommendation

Approaches such as mini-batch sampling, parameter-efficient adapters, and approximate nearest-neighbour indices help—but no silver bullet exists. (see Table 1.10)

1.6.4 Privacy & Security

- **Location privacy.** Four GPS points can re-identify individuals.
- **Regulatory compliance.** GDPR/CCPA restrict retention and mandate purpose limitation.
- **Attack surface.** Model inversion or membership inference may expose sensitive visits (clinics, religious sites).

Emerging defences: on-device inference, federated learning, differential-privacy noise, synthetic trajectory generation—each trading accuracy for confidentiality.

1.6.5 Fairness & Bias

Different bias sources and their effect are explained in table 1.11:
Open question: How to inject equity constraints (group coverage, geographic fairness) into sequential, spatially-aware recommenders without crippling relevance?

Bias source	Potential harm	Diagnostic metric
Data imbalance	Affluent districts receive better rankings → "service deserts''.	Disparate impact, coverage ratio
Feedback loops	Popular venues amplified; minority-owned businesses stagnate.	Popularity lift vs. baseline traffic
Temporal censoring	Curfews penalise night-shift workers.	Time-segment parity

Table 1.11: Fairness hazards in geospatial recommendation

1.6.6 Interpretability & User Trust

- **Black-box opacity.** Dynamic graph models mix message passing, attention, and recurrent memory—hard to trace individual recommendations.
- **Explanatory demand.** Regulators and users increasingly ask "Why this place?''
- **Possible avenues.** Counterfactual saliency maps, path-based explanations ("because you often visit organic cafés near parks''), attention visualisation in space-time.

Conclusion

By grounding spatio-temporal recommendation in fundamental learning paradigms, network architectures, and robust metrics, we've created a clear framework for understanding both opportunities and pitfalls. This foundation ensures that subsequent, more specialized methods can be compared on equal footing, and it highlights the key challenges—cold-start, sparsity, shifting distributions—that any real-world recommender must overcome.

Chapter 2

Graph–Centric Representation Learning

Introduction

Graphs offer a powerful lens for capturing rich, multi-relation knowledge—linking users, items, locations, and time in a single structure. In this chapter, we demystify graph theory and knowledge graphs, then explore how to translate that symbolic structure into numerical embeddings and how to build neural models that can reason over static and evolving graphs. We conclude with a conceptual map of how graphs can elevate recommendation beyond simple sequences or matrices.

2.1 Essentials of Graph Theory

Spatio-temporal recommender systems enriched with knowledge graphs ultimately manipulate *graphs*—mathematical objects that encode entities and their relationships. A firm grasp of core graph-theoretic concepts therefore underpins every representation-learning technique introduced later in upcoming chapters.[6]

2.1.1 Graphs, Nodes and Edges

A graph is denoted $G = (V, E)$, where

- $V = \{v_1, \ldots, v_{|V|}\}$ is the set of *nodes / vertices* (users, POIs, street intersections, time slots);
- $E \subseteq V \times V$ is the set of *edges* capturing pairwise relations.

The graph can be represented by its adjacency matrix $A \in \{0, 1\}^{|V| \times |V|}$ with $A_{ij} = 1$ iff $(v_i, v_j) \in E$.

Graph type	Extra structure	Notation / storage	Mobility example
Weighted	Edge weight $w : E \to \mathbb{R}_{>0}$ encodes intensity, distance, cost	Sparse matrix, CSR	Driving time between road nodes
Attributed / property	Node features $\mathbf{x}_v$; edge features $\mathbf{e}_{uv}$	Key–value maps	POI category, road type
Multi-relational (hetero)	Edge label $\rho : E \to \mathcal{R}$ with $\lvert\mathcal{R}\rvert > 1$	Triple store / RDF	`located_in`, `next_to`
Temporal / dynamic	Time-stamped edges E_t or events (u, v, τ)	Event log, snapshots	Check-in at 12:03, 18 May 2025

Table 2.1: Richer graph flavours beyond the unweighted, static case

2.1.2 Beyond Simple Graphs: Richer Edge Semantics

Knowledge graphs combine multi-relational and attributed aspects, whereas temporal graphs are indispensable when interactions evolve (*e.g.*, check-ins, traffic flows). See table 2.1

2.1.3 Walks, Paths, Trails and Cycles

Explained in table 2.2:

Concept	Definition	Relevance
Walk	Sequence $v_0, v_1, \ldots, v_k$ with $(v_{i-1}, v_i) \in E$	Random-walk embeddings, PageRank
Trail	Walk with no repeated *edges*	Route planning without edge reuse
Path	Walk with no repeated *nodes*	Shortest-path distance, connectivity proofs
Cycle	Closed trail ($v_0 = v_k$)	Community motifs; negative-cycle detection

Table 2.2: Traversal notions and their uses

Shortest-path algorithms (Dijkstra, A*) rely on edge weights; random walks underpin DeepWalk, Node2Vec, and influence propagation.

2.1.4 Connectivity Notions

- **Strongly connected component (SCC)**: in a directed graph, every node is reachable from every other via a directed path.
- **Weakly connected component (WCC)**: connectivity after ignoring edge directions.

- k**-core**: maximal subgraph where each node has degree $\geq k$; useful for pruning sparse KGs before learning.

Connectivity guides mini-batch sampling and cold-start heuristics: isolated nodes lack structural context.

2.1.5 Communities and Modular Structure

Real-world graphs exhibit densely linked *communities* with sparse external ties. (See table 2.3)

Method	**Idea**	**Complexity**	**Suitability**
Louvain (modularity)	Greedy merges maximise modularity	$\mathcal{O}(\|E\|)$	Large static graphs
Spectral clustering	Laplacian eigen-vectors partition nodes	$\mathcal{O}(\|V\|^3)$	Medium, dense graphs
Label propagation	Nodes adopt majority neighbour label	Near-linear	Dynamic graphs, speed critical
Infomap	Compress random-walk description length	$\mathcal{O}(\|E\|)$	Hierarchical communities

Table 2.3: Popular community-detection algorithms

Community membership can supply priors for embeddings (Section 2.3) and fairness auditing (Section 1.6).

2.2 From Graphs to Vectors: Embedding Methods

In order to leverage graph-structured information in downstream machine learning (ML) tasks—such as recommendation, classification, or clustering—it is necessary to transform symbolic graph entities (nodes, edges, and relations) into numeric representations. Graph embeddings accomplish this by mapping each node (and, in some cases, edges or entire subgraphs) to a low-dimensional vector in $\mathbb{R}^d$. These continuous representations preserve key properties of the original graph—structural proximity, semantic relationships, and the ability to generalize to unseen nodes or relations. This section outlines (1) the primary goals of graph embeddings, (2) the main families of embedding algorithms, and (3) the common training objectives (loss formulations).[3]

2.2.1 Goals of Embeddings

Graph-specific embeddings aim to satisfy three interrelated criteria:

Structural Proximity Embeddings reflect graph topology so that nodes which are closely connected remain close in embedding space. For example, nodes sharing common neighbors should have embeddings with high cosine similarity.

Semantic Coherence Embeddings encode the semantic meaning of relations and attributes, such that vector operations capture semantic roles accurately.

Inductive Generalization Embeddings can infer vectors for unseen nodes or relations based on learned patterns from known data.

2.2.2 High-Level Families of Embedding Algorithms

Embedding methods fall into four primary categories: translational, factorization-based, random-walk-based, and neural encoder-based methods.

Translational Models

Translational models treat each triple (h, r, t) as a vector relation:

$$h + r \approx t. \tag{2.1}$$

For instance, TransE enforces:

$$|h + r - t|_2 \quad \text{is small for true triples, and large otherwise.} \tag{2.2}$$

Variants include TransH, TransR, and TransD, which introduce relation-specific projections.

Matrix and Tensor Factorization

These methods factorize adjacency matrices or tensors representing multi-relational graphs. For example, RESCAL decomposes each relational adjacency matrix X_r as:

$$X_r \approx HR_rH^{\top}, \tag{2.3}$$

DistMult simplifies this to diagonal relation matrices, with triple scores computed by:

$$\langle h, r, t\rangle = \sum_i = 1^d h_ir_it_i. \tag{2.4}$$

Random-Walk-Based Methods

These methods generate node sequences via random walks and employ word2vec-style skip-gram optimization to learn embeddings. Leading examples include DeepWalk and Node2Vec, which capture higher-order proximity implicitly.[21]

Neural Encoder-Based Methods (Graph Neural Networks)

Graph neural networks (GNNs) apply message passing:

$$\text{Message computation: } m_u^{(\ell)} = \text{AGGREGATE}(h_v^{(\ell-1)} | v \in \mathcal{N}(u)), \tag{2.5}$$

$$\text{Update: } h_u^{(\ell)} = \text{UPDATE}(h_u^{(\ell-1)}, m_u^{(\ell)}). \tag{2.6}$$

These methods inherently support inductive learning and incorporate node attributes effectively.

2.2.3 Training Objectives

Embedding methods optimize objectives that enforce desirable geometric relationships. Two main categories are margin-based ranking losses and probabilistic losses.

Margin-Based Ranking Losses

Given a triple (h, r, t) and corrupted triples (h', r, t'), the margin loss is:

$$L = \sum_(h, r, t) \in D^+ \sum_(h', r, t') \in D^- \max(0, \gamma + s(h, r, t) - s(h', r, t')), \tag{2.7}$$

where $\gamma > 0$ is a margin hyperparameter. For instance, TransE uses:

$$s(h, r, t) = |h + r - t|_2. \tag{2.8}$$

Probabilistic Losses (Softmax / Cross-Entropy)

These methods define a probability distribution for node-context pairs, optimized by negative log-likelihood:

$$p(v|u) = \frac{\exp(u^\top v)}{\sum_v' \in V \exp(u^\top v')}, \tag{2.9}$$

with a loss:

$$L = -\sum_{(u,v)\in C} \log p(v|u). \tag{2.10}$$

To improve computational efficiency, negative sampling approximates this:

$$\log \sigma(u^\top v) + \sum_i = 1^K \mathbb{E}_v_i \sim P_n(v) \; [\log \sigma(-u^\top v_i)]. \tag{2.11}$$

GNN-based link prediction often uses binary cross-entropy:

$$L = -\sum_(u, v) \in D^+ \log \sigma(\phi(u, v)) - \sum_(u', v') \in D^- \log(1 - \sigma(\phi(u', v'))), \tag{2.12}$$

where $\phi(u, v)$ is a scoring function predicting edge existence. Negative samples are typically drawn uniformly or using heuristics.

2.3 Graph Machine Learning

Graph machine learning encompasses methods that operate on graph-structured data, learning to propagate, aggregate, and transform information along edges rather than treating nodes as independent feature vectors. In recommender systems, these methods exploit multi-hop relationships—such as user–item–category interactions, social influences, and temporal trajectories—by iteratively exchanging "messages" across a graph. This section surveys (1) the canonical message-passing neural network (MPNN) paradigm and its common variants, (2) the distinction between static and dynamic (temporal) graphs and how temporal message passing is performed, and (3) key considerations around expressiveness, over-smoothing, and scalability.[22]

2.3.1 Message-Passing Neural Networks (GNNs) and Variants

The General MPNN Framework

Message-passing neural networks (often referred to as graph neural networks, or GNNs) learn node representations by repeatedly exchanging information along edges. Concretely, a generic MPNN layer indexed by ℓ operates as follows for each node u in graph $G = (V, E)$:

Message Computation Each neighbor $v \in \mathcal{N}(u)$ sends a "message" $m_{v\to u}^{(\ell)}$. Typically, this message is some learnable function of the neighbor's previous embedding $h_v^{(\ell-1)}$ and potentially the edge feature $e_{v,u}$:

$$m_{v\to u}^{(\ell)} = \mathrm{Msg}^{(\ell)}\Big(h_v^{(\ell-1)},\, e_{v,u}\Big). \tag{2.13}$$

Aggregation Node u aggregates all incoming messages from its neighbors via a permutation-invariant function (e.g., sum, mean, or max):

$$a_u^{(\ell)} = \bigoplus_{v\in\mathcal{N}(u)} m_{v\to u}^{(\ell)}, \tag{2.14}$$

where $\bigoplus$ denotes the chosen aggregation operator. The goal is to summarize local neighborhood information in a single vector $a_u^{(\ell)}$.

Update Finally, u updates its embedding by combining its previous embedding $h_u^{(\ell-1)}$ with the aggregated message $a_u^{(\ell)}$ through a learnable transformation (often a linear layer plus nonlinearity):

$$h_u^{(\ell)} = \mathrm{Upd}^{(\ell)}\Big(h_u^{(\ell-1)},\, a_u^{(\ell)}\Big). \tag{2.15}$$

After L layers of message passing, $h_u^{(L)}$ encodes information from all nodes up to L hops away (cf. equations (2.13)–(2.15)).

At initialization ($\ell = 0$), each node u is assigned an input feature vector $h_u^{(0)}$. If no explicit features exist, one may use learnable "node-ID" embeddings or simple one-hot indicators. Edge features $e_{v,u}$—such as relation types, timestamps, or weights—can be incorporated into the message function (2.13) to explicitly distinguish different connections.

Common GNN Variants

A variety of GNN architectures instantiate the MPNN framework by specifying different message, aggregation, and update functions. Some canonical models include:

Graph Convolutional Network (GCN) (Kipf & Welling, 2017)

- **Message:** For each neighbor v, compute

$$\frac{1}{\sqrt{\deg(u)\,\deg(v)}}\, h_v^{(\ell-1)}.$$

- **Aggregation:** Sum over normalized neighbor embeddings.
- **Update:** Apply a shared weight matrix $W^{(\ell)}$ and ReLU:

$$h_u^{(\ell)} = \mathrm{ReLU}\left(\sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{1}{\sqrt{\deg(u)\,\deg(v)}}\, h_v^{(\ell-1)}\, W^{(\ell)} \right). \tag{2.16}$$

This formulation can be interpreted as a spectral convolution on graphs (using a first-order Chebyshev approximation), effectively mixing self-information with neighbor information at each layer (see equation (2.16)).

GraphSAGE (Hamilton *et al.*, 2017) Designed for inductive settings, GraphSAGE learns an aggregator (mean, LSTM, or max-pool) that combines neighbor embeddings. For the "mean" variant:

$$h_u^{(\ell)} = \sigma\Big(W^{(\ell)} \left[h_u^{(\ell-1)} \parallel \mathrm{mean}_{v \in \mathcal{N}(u)}\, h_v^{(\ell-1)} \right] \Big), \tag{2.17}$$

where $\parallel$ denotes concatenation and σ an activation (e.g., ReLU). GraphSAGE employs neighborhood sampling (sampling a fixed number of neighbors per node) to control computational cost, enabling scalability to large graphs and on-the-fly embedding of new nodes (given their features and a list of neighbors), as shown in equation (2.17).

Graph Attention Network (GAT) (Velickovic *et al.*, 2018) Introduces an attention mechanism to weight neighbor messages differently:

$$\alpha_{uv}^{(\ell)} = \frac{\exp\Big(\mathrm{LeakyReLU}\Big(a^\top \left[W\, h_u^{(\ell-1)} \parallel W\, h_v^{(\ell-1)} \right]\Big)\Big)}{\sum_{w \in \mathcal{N}(u)} \exp\Big(\mathrm{LeakyReLU}\Big(a^\top \left[W\, h_u^{(\ell-1)} \parallel W\, h_w^{(\ell-1)} \right]\Big)\Big)}, \tag{2.18}$$

$$h_u^{(\ell)} = \sigma\Big(\sum_{v \in \mathcal{N}(u)} \alpha_{uv}^{(\ell)}\, W\, h_v^{(\ell-1)} \Big). \tag{2.19}$$

By learning attention coefficients $\alpha_{uv}^{(\ell)}$, GAT can focus on the most informative neighbors—crucial in heterogeneous or noisy graphs (see equations (2.18)–(2.19)).

Graph Isomorphism Network (GIN) (Xu *et al.*, 2019) Demonstrates that simple, injective aggregation (sum followed by MLP) can achieve maximal discriminative power equivalent to the Weisfeiler-Lehman test:

$$h_u^{(\ell)} = \mathrm{MLP}^{(\ell)}\left(\left(1 + \epsilon^{(\ell)}\right) h_u^{(\ell-1)} \; + \sum_{v \in \mathcal{N}(u)} h_v^{(\ell-1)} \right), \tag{2.20}$$

where $\epsilon^{(\ell)}$ is learnable or fixed. GINs excel at capturing subtle structural differences but—like most GNNs—can overfit on small graphs (see (2.20)).

Relational GNNs (R-GNNs) For multi-relational KGs with typed edges, R-GNNs (e.g., R-GCN by Schlichtkrull *et al.*, 2018) assign a distinct weight matrix $W_r^{(\ell)}$ for each relation type r. The update becomes:

$$h_u^{(\ell)} = \sigma\left(\sum_{r\in\mathcal{R}} \sum_{v\in\mathcal{N}_r(u)} \frac{1}{c_{u,r}} W_r^{(\ell)} h_v^{(\ell-1)} \; + \; W_0^{(\ell)} h_u^{(\ell-1)}\right), \tag{2.21}$$

where $\mathcal{N}_r(u)$ are neighbors connected by relation r, and $c_{u,r}$ is a normalization constant. By explicitly modeling relation types, R-GNNs can differentiate "actedIn" from "directedBy," crucial for knowledge-enhanced recommendations (see (2.21)).

Collectively, these variants share the core idea of iterative neighborhood aggregation but differ in how they compute messages (e.g., linear transform vs. attention), aggregate (mean vs. sum vs. max vs. learned weighting), and update node representations (simple linear layer vs. multi-layer perceptron). The choice among them depends on the application context, graph size, and the importance of inductive generalization.

2.3.2 Static vs. Dynamic Graphs; Temporal Message Passing

Real-world recommender systems often operate on graphs that evolve over time—new users join, items are introduced, interactions occur at specific timestamps. Accordingly, graph machine learning must address both static and dynamic (temporal) settings.[10]

Static Graphs

A static graph $G = (V, E)$ is fixed during both training and inference. All node/edge additions or deletions are aggregated into a single snapshot. Most foundational GNN research (e.g., GCN, GraphSAGE, GAT) assumes static graphs, where neighborhood structure does not change. In recommendation, one might build a bipartite graph of users and items—aggregating all historical interactions—and train a static GNN to learn embeddings. While conceptually simple, this "frozen snapshot" approach ignores fine-grained temporal order and cannot easily accommodate new nodes at inference time (unless using inductive variants like GraphSAGE with user/item features).

Dynamic (Temporal) Graphs

Dynamic graphs model time explicitly. Two principal formulations exist:

Discrete-Time Temporal Graphs (Snapshot Sequence) The timeline is divided into T discrete intervals. At each time t, one observes a graph snapshot $G(t) = (V(t), E(t))$. For example, $G(1)$ might contain all interactions up to January, $G(2)$ those up to February, and so on. Temporal GNNs on discrete snapshots either (a) train a separate model per snapshot (inefficient) or (b) propagate node states forward through time—e.g., through a recurrent module that takes node embeddings at $t-1$ and updates them using the structure of $G(t)$.

EvolveGCN (Pareja *et al.*, 2019) uses an RNN to evolve GCN weights over successive snapshots. Recurrent GNNs couple a GNN layer with an LSTM (or GRU) per node: at time

t, node u computes a GNN update based on $G(t)$ and then feeds the resulting embedding to a GRU cell that maintains memory across time.

Continuous-Time Temporal Graphs Instead of chunking the timeline, continuous-time approaches treat each edge as occurring at a precise timestamp t_e. Models process events sequentially and update only the affected nodes[17]. Notable examples:

- **Temporal Graph Networks (TGNs)** (Rossi *et al.*, 2020) maintain a memory module per node. Upon each interaction event $(u, v, t_e, \text{features})$, they (a) retrieve the current memory of u and v, (b) compute a message using a message-function MLP that takes node memories and edge features, (c) update memories via a memory-unit (e.g., GRU), and (d) optionally recompute embeddings through a small GNN that aggregates from neighbors whose edges occurred before t_e. TGNs thus operate directly in continuous time and naturally scale to streaming event data.
- **Graph Attention Temporal Convolutional Networks (GAT-TCN)** and other variants apply temporal convolution or attention over sequences of neighbor embeddings, effectively learning both structural and temporal patterns.

Dynamic models capture evolving user preferences and item popularity. For instance, if a user's interactions shift from "action movies" to "documentaries" over time, a temporal GNN can adapt that user's embedding to reflect recent trends, improving recommendation recency and relevance. However, dynamic modeling introduces complexity—one must decide how to discretize time (if at all), maintain node memories efficiently, and sample past interactions without incurring large memory overhead.

2.3.3 Expressiveness, Over-Smoothing, and Scalability

As GNNs proliferate, three core challenges have emerged: expressiveness (how well they distinguish graph structures), over-smoothing (loss of feature diversity in deep networks), and scalability (handling large graphs).[9]

Expressiveness

A common question is: "What classes of graph functions or structures can a given GNN distinguish?" The Weisfeiler-Lehman (WL) test offers a theoretical lens: if two non-isomorphic graphs cannot be distinguished by the 1-dimensional WL algorithm, then any GNN that aggregates neighbors by simple sum/mean cannot distinguish them either.

The Graph Isomorphism Network (GIN) introduced an injective aggregation (sum + MLP) that matches the discriminative power of the WL test (see equation (2.20)), making it maximally "expressive" under that paradigm. Limitations arise with regular graphs or cospectral graphs—injective aggregators and higher-order message passing can partially alleviate these issues but often at the cost of efficiency.

In practice, a balance must be struck: while highly expressive GNNs can capture fine-grained structural differences, many recommendation tasks benefit from simpler models that capture community structure and attribute homophily. Overly expressive models can overfit on noisy graphs, especially when labeled data are scarce.

Over-Smoothing

As GNN depth increases, repeated neighborhood mixing can cause node embeddings to converge to similar or identical vectors—a phenomenon known as over-smoothing. When all node representations become indistinguishable, the network loses discriminative power. Empirical observations show that standard GCNs often reach peak performance with only 2–3 layers; beyond that, accuracy can drop. Common mitigation strategies include:

- **Residual (Skip) Connections**: Each layer incorporates a fraction of its input:

$$h_u^{(\ell)} = \sigma\Big(W^{(\ell)}\,\mathrm{AGG}(\mathcal{N}(u))\Big) \;+\; h_u^{(\ell-1)}. \tag{2.22}$$

- **Normalization (BatchNorm/LayerNorm)**: Helps preserve variance across layers.
- **Jumping Knowledge (JK) Networks** (Xu *et al.*, 2018): Concatenate or max-pool embeddings from all intermediate layers, allowing the model to select the "best" neighborhood radius for each node.
- **DropEdge** (Rong *et al.*, 2020): Randomly removing a fraction of edges during training reduces excessive neighborhood mixing.
- **PairNorm** (Zhao & Akoglu, 2019): Normalizes pairwise distances to prevent embedding collapse.

These techniques help preserve meaningful feature differences, especially when deeper GNNs are desired to capture long-range dependencies (e.g., interactions several hops away).

Scalability

Large-scale recommender graphs can contain millions of users and items, with tens or hundreds of millions of edges. Directly applying naïve GNNs—where each layer aggregates messages from all neighbors—becomes infeasible. Principal strategies to achieve scalability include:

Neighborhood Sampling Rather than aggregating over an entire neighborhood $\mathcal{N}(u)$, sample a fixed number S of neighbors per node at each layer. For example, GraphSAGE's Sample–Aggregate method uniformly samples S_ℓ neighbors at layer ℓ, limiting computation per node to $O(S_\ell)$.

Graph Partitioning and Mini-Batching Partition the large graph into smaller subgraphs or clusters, then train the GNN on one subgraph at a time (e.g., Cluster-GCN by Chiang *et al.*, 2019). By ensuring that most edges remain within clusters, one can approximate full-graph training while drastically reducing memory footprint.

Precomputing Sparse Matrix Multiplications For static graphs, precompute normalized adjacency matrix powers or store sparse factorizations to accelerate GCN-style propagation. However, this approach loses flexibility when node features change or for dynamic graphs.

Edge-Centric and Sampling-Free Methods Recent research explores training directly on edges (e.g., GraphSAINT, Zeng *et al.*, 2020), sampling subgraphs based on edge distributions rather than nodes to reduce variance and achieve faster convergence.

Distributed and GPU-Accelerated Training Leveraging multi-GPU clusters or CPU-GPU hybrid architectures allows large graphs to be partitioned across machines. Frameworks such as DGL and PyTorch Geometric implement partitioning and message passing that span multiple devices.

Selecting an appropriate scalability strategy depends on graph density, memory constraints, and desired throughput. For recommender systems, techniques supporting incremental updates or continuous training (e.g., streaming TGNs) are particularly valuable.

2.4 Graph-Enhanced Recommendation: A Conceptual View

Recommender systems traditionally rely on user–item interaction matrices or sequential logs of behavior (e.g., clickstreams, purchase histories). While effective in many settings, these paradigms often overlook complex, multi-hop relationships and rich heterogeneity present in modern platforms—such as user social ties, item attributes, spatio-temporal context, and complementary side information (e.g., tags, reviews, categories). By integrating a graph-centric perspective, recommender models can explicitly leverage relational patterns that span multiple hops, accommodate diverse node and edge types, and capture latent structures that purely sequence- or matrix-based methods cannot. In this section, we present a conceptual overview of how graph representations address three core needs: (1) capturing multi-hop relational context, (2) handling heterogeneity via typed edges and nodes, and (3) outlining the advantages of graph-enhanced approaches relative to sequence- and matrix-based baselines.[22]

2.4.1 Capturing Multi-Hop Relational Context

A fundamental limitation of traditional collaborative filtering (CF) matrix factorization is its reliance on direct user–item co-occurrence. If a user u has interacted only with items $\{i_1, i_2\}$, a pure matrix-factorization model can infer similarity between u and other users who share those exact interactions, but it struggles to exploit indirect connections—such as users who liked i_1 and some related item i_3 that u has not yet discovered. Graph representations naturally encode these indirect signals:

- **User → Item → Attribute → Item → … chains.** Suppose the platform knows that
  1. u purchased item i_1.
  2. i_1 belongs to category "Outdoor Gear."
  3. Another item i_3 also belongs to "Outdoor Gear."

A multi-hop path

$$u \xrightarrow{\text{purchased}} i_1 \xrightarrow{\text{hasCategory}} \text{Outdoor Gear} \xrightarrow{\text{categoryOf}} i_3$$

signals that i_3 is semantically related to u even if u never directly interacted with i_3. A graph-enhanced recommender can exploit this path—possibly weighting it by edge confidences or path length—to surface i_3 as a candidate recommendation.

- **User → Friend → Item.** In a social recommendation scenario, if user u has no direct purchases in category "Camping Tents" but their friend v frequently buys tents, a graph path

  $$u \xrightarrow{\text{follows}} v \xrightarrow{\text{purchased}} i_{\text{tent}}$$

  provides an implicit signal that i_{tent} may interest u. Sequence-based models (operating on u's own click or purchase sequence) cannot capture this socially mediated signal unless they explicitly aggregate friends' behaviors. A graph approach unifies these interactions under one structural framework.

- **Knowledge-Driven Inference.** When a knowledge graph contains richer semantics—e.g., movie metadata—multi-hop paths can traverse genre hierarchies or cast networks. For instance,

  $$u \xrightarrow{\text{liked}} \text{Movie A} \xrightarrow{\text{directedBy}} \text{Director X} \xrightarrow{\text{directedBy}^{-1}} \text{Movie B.}$$

  Even if u has not watched Movie B, the chain through a shared director strongly implies preference. Graph-enhanced recommenders can learn to aggregate—and even weight—distinct path types, yielding explainable suggestions: "Because you liked Movie A by Director X, you might also like Movie B."

By encoding these multi-hop relationships explicitly in a graph, embedding or message-passing methods can learn representations whereby distance (or similarity) in the embedding space reflects not only co-occurrence but also semantically meaningful relational context. In practice, this reduces data sparsity: even if a user has few direct interactions, the rich neighborhood structure can "fill in" missing information through attribute- and relation-driven propagation.

2.4.2 Handling Heterogeneity Through Typed Edges and Nodes

Modern platforms involve heterogeneous entities—users, items, categories, tags, points-of-interest (POIs), contextual nodes (e.g., time slots or locations), and social profiles—and myriad relationship types among them. A homogeneous representation (treating all nodes identically) or collapsing all relations into a single "edge" can obscure critical distinctions. Graph-enhanced recommenders address this by:

Typed Nodes

- **User nodes** (u_i), annotated with demographic attributes, social features, or behavioral profiles.
- **Item nodes** (i_j), characterized by content attributes (e.g., genre, price, brand, geocoordinates).
- **Auxiliary nodes** (e.g., *Category:Electronics*, *Brand:Sony*, *TimeSlot:Morning*, *Location:Paris*).

Each node type may carry its own feature vector (e.g., user embedding from profile data, item embedding from textual description) or an ID embedding when explicit features are unavailable.

Typed Edges (Relations)

- **Interaction edges** (e.g., *purchased*, *viewed*, *clicked*, *rated*). Each can carry a weight or timestamp.
- **Attribute edges** (e.g., *hasCategory*, *hasBrand*, *locatedIn*). These are often many-to-many (an item can have multiple categories or a user multiple locations).
- **Social edges** (e.g., *follows*, *friendsWith*, *trusts*).
- **Temporal or Contextual edges** linking a user or item to a time node (*accessedAt*, *viewedAt*).
- **Co-occurrence edges** linking items that appear together in a basket or session (*coPurchasedWith*).

By preserving each relation's type, graph neural networks (particularly relational GNNs) can learn relation-specific transformations. For instance, the aggregation rule for *purchased* edges (capturing explicit preference) might differ from that for *viewedAtTimeSlot* edges (capturing temporal browsing habits). R-GNNs assign distinct weight matrices (or attention heads) per relation, enabling the model to calibrate the influence of heterogeneous signals.

Example: Incorporating Spatio-Temporal Context A location-aware POI recommender might build a graph wherein:

- A user u is connected via *visitedAt* edges to specific time-slot nodes (e.g., *Monday_18:00*).
- POI nodes exhibit *locatedIn* edges to region nodes (e.g., *Le Marais*).
- Each POI has *hasCategory* edges (museum, cafe, park).
- Additionally, a time window node (e.g., *Summer_2024*) captures seasonality.

The resulting heterogeneous graph merges user spatial-temporal preferences with POI attributes. A GNN can propagate signals along typed edges—e.g., "aggregate user's morning visits in Le Marais" to boost recommendations of morning-friendly cafes in that region—whereas a matrix approach would need hand-crafted cross-features (e.g., user $\times$ location $\times$ time slot) that quickly explode combinatorially.

Modeling Multi-Modal Side Information Beyond structural heterogeneity, graph-enhanced approaches facilitate seamless integration of side information:

- **Textual embeddings** from item descriptions or user reviews can be attached as initial node features.
- **Visual features** (e.g., CNN embeddings of product images) can augment item nodes.
- **Geospatial coordinates** can produce continuous node attributes or be discretized into region nodes.

A purely matrix-factorization model would struggle to unify modalities; typical workarounds involve concatenating latent factors with side-feature vectors or designing separate pipelines, risking misalignment. In a graph, these heterogeneous modalities co-exist as different node types (or features), and GNN-based message passing naturally blends them according to learned weights.

2.4.3 Advantages over Purely Sequence- or Matrix-Based Approaches

While session-based and sequence-aware recommenders (e.g., using RNNs, Transformers, or Markov Chains) model the order of user interactions, they still often treat items as atomic tokens with no explicit relational structure. Matrix-factorization approaches capture latent similarities but cannot exploit multi-hop paths or side information without extensive feature engineering. Graph-enhanced recommendation offers several advantages:

Explicit Structural Reasoning

- Sequence models focus on next-item prediction given a sliding window of past interactions. They capture short-term sequential dependencies but typically do not incorporate attribute or social signals unless explicitly concatenated at each time step.
- Matrix-factorization captures global co-occurrence patterns but ignores order, context, and side-information. A user who watches three science-fiction movies in a row and then shifts to drama cannot signal sequence transitions unless the model is retrained.
- Graph models unify all signals—sequential, co-occurrence, attribute, social—within one structure. Message passing can propagate not only along direct user–item edges but also along category hierarchies, social ties, and temporal links, capturing hybrid patterns that neither sequence nor matrix alone can represent.

Better Data Efficiency and Cold-Start Handling

- New items or users with no interaction history remain a major challenge for both matrix and sequence models (the "cold-start" problem). Without explicit features, these methods cannot produce reliable recommendations.

- Graph approaches can leverage side information even when interactions are sparse. For example, a newly introduced product node connected to attribute nodes (*Brand:Sony*, *Category:Electronics*) can inherit embedding context from existing Sony products or electronics subgraph. A user node with no purchase history but known demographic attributes (e.g., *Age:25–34*, *Location:Berlin*) can connect to corresponding attribute nodes, allowing the GNN to position that user within an existing attribute subspace. Consequently, cold-start coverage improves.

Explainability via Paths and Subgraphs

- Sequence models offer implicit attention weights (e.g., self-attention scores), but human-readable explanations often require post-hoc analysis.
- Matrix methods can generate "nearest-neighbor" lists in latent space, but these lack intuitive links to real-world attributes.
- In graph-enhanced recommendation, one can trace which multi-hop paths contributed most to a predicted score. For instance, a recommendation can be explained as:

  "Because you liked items that share director *Quentin Tarantino*, and you follow friends who also rated *Pulp Fiction* highly."

  Such path-based explanations are naturally supported by the graph structure.

Unified Treatment of Contextual and Social Dimensions

- Sequence models struggle to incorporate social or context features without elaborate architectures (e.g., hierarchical RNNs for social influences, or joint session/context encoders).
- Matrix-factorization variants can incorporate side information via factorization machines or hybrid extensions, but these typically require manually specifying feature crosses.
- Graph models treat social relations (e.g., *follows*), contextual factors (e.g., time, location), and item attributes (e.g., category) as edges or auxiliary nodes. A single GNN-based encoder can simultaneously learn from all these factors by customizing message-passing rules per relation type.

Flexible Aggregation and Attention Mechanisms

- Sequence models attend over recent history, but their notion of "neighbor" is limited to linear ordering.
- Graph models with attention (e.g., GAT) can learn to weigh different neighbors differently—even across relation types—enabling, for example, discounting spurious category links when computing user embeddings.
- This learned weighting is more expressive than a fixed sequence window or uniform neighbor sampling in matrix baselines.

2.5 Related Work

Our methodology synthesizes advances across three complementary domains:

Urban Knowledge Graphs The UUKG dataset provides a comprehensive, open-source UrbanKG that unifies administrative boundaries, POIs, road networks, and user trajectories into a single graph structure. Its generalized schema and large scale form the backbone for any knowledge-driven urban forecasting task.[15]

Knowledge-Graph Embedding To transform symbolic graphs into trainable features, translational methods like TransE (Work 3)[3] embed entities and relations in Euclidean space, while recent manifold-aware models like GIE (Work 2)[4] leverage mixed-curvature geometries to better capture hierarchical and cyclic patterns. Both approaches have demonstrated strong link-prediction performance on urban triples, encoding spatial hierarchies in compact vectors.

Spatio-Temporal Prediction with Graph Embeddings Embedding vectors from UrbanKGs have been used to enrich traditional forecasting models—concatenated with time-series features or plugged into static GNN architectures (e.g. STGCN, ASTGCN). However, human mobility unfolds on continually evolving graphs, calling for dynamic graph neural methods.

Model	**Mem.**	**Mem. Updater**	**Embedding**	**Mess. Agg.**	**Mess. Func.**
Jodie	node	RNN	time	—†	id
TGAT	—	—	attn (2ℓ, 20n)*	—	—
DyRep	node	RNN	id	—‡	attn‖
TGN-attn	node	GRU	attn (1ℓ, 10n)	last	id
TGN-2ℓ	node	GRU	attn (2ℓ, 10n)	last	id
TGN-no-mem	—	—	attn (1ℓ, 10n)	—	—
TGN-time	node	GRU	time	last	id
TGN-id	node	GRU	id	last	id
TGN-sum	node	GRU	sum (1ℓ, 10n)	last	id
TGN-mean	node	GRU	attn (1ℓ, 10n)	mean	id

Table 2.4: Previous models for deep learning on continuous-time dynamic graphs are specific case of our TGN framework. Shown are multiple variants of TGN used in our ablation studies. method (l,n) refers to graph convolution using l layers and n neighbors. yuses t-batches. $*$ uses uniform sampling of neighbors, while the default is sampling the most recent neighbors. zmessage aggregation not explained in the paper. k uses a summary of the destination node neighborhood (obtained through graph attention) as additional input to the message function.

Dynamic Graph Neural Networks Temporal Graph Networks (TGN, Work 4)[16] introduce per-node memory modules that accumulate interaction history, while TGAT, which is considered as a variant of the TGN framework (see table 2.4), applies multi-head attention over a node's recent temporal neighbors without persistent memory. Algorithm 1 presents the pseudocode for TGN training.

Algorithm 1 Training TGN

```
s ← 0                                                   ▷ Initialize memory summary to zeros
m_raw ← {}                ▷ Initialize raw message buffer batch (i, j, e, t) in training data
n ← sample negatives
m ← msg(m_raw)                                          ▷ Compute messages from raw features
m̄ ← agg(m)                                             ▷ Aggregate messages for the same nodes
ŝ ← mem(m̄, s)                                          ▷ Get updated memory
z_i, z_j, z_n ← emb_s(i, t), emb_s(j, t), emb_s(n, t)   ▷ Compute node embeddings
P_pos, P_neg ← dec(z_i, z_j), dec(z_i, z_n)             ▷ Compute interaction probabilities
ℓ ← BCE(P_pos, P_neg)                                   ▷ Compute BCE loss
m_raw, m_r̃aw ← (ŝ_i, ŝ_j, t, e), (ŝ_j, ŝ_i, t, e)       ▷ Compute raw messages
m_raw ← store_raw_messages(m_raw, m_r̃aw)                ▷ Store raw messages
s_i, s_j ← ŝ_i, ŝ_j          ▷ Store updated memory for sources and destinations
```

JODIE (Work 5)[12] uses coupled recurrent units to co-evolve user and item embeddings event by event. These frameworks form our evaluation suite for comparing how KG embeddings aid sequence-based, attention-based, and memory-based mobility predictors.

2.6 Evaluation & Benchmarking in Graph ML

Building on the general evaluation methodology outlined in Section 1.5, graph-centric machine learning requires specialized protocols and metrics to assess performance on tasks such as link prediction, node classification, and temporal link forecasting. Moreover, characteristics intrinsic to many real-world graphs—sparsity of connections, skewed degree distributions, and continuous evolution—directly affect how we sample train/validation/test splits, choose negative examples, and interpret quantitative results. This section discusses (1) core tasks and their associated evaluation criteria, (2) strategies for temporal forecasting and hold-out in dynamic graphs, and (3) the impact of graph sparsity and updates on reliability and comparability of metrics.[20]

2.6.1 Link Prediction and Node Classification

Link Prediction

Definition. In link prediction, the goal is to infer missing edges (or relations) between nodes, given an incomplete snapshot of the graph. In recommendation settings, this is analogous to inferring which user–item pairs are likely to be observed in the future, with the graph defined by existing interactions or affinities.

Data Splitting.

- **Masking Edges:** Randomly remove a subset of existing edges (hold-out set E_{test}) from the observed training graph G_{train}. The remaining edges E_{train} form the positive training set.
- **Negative Sampling:** For every removed (positive) edge $(u, v) \in E_{\text{test}}$, sample a set of non-existing node pairs $(u', v') \notin E$ as negative examples. It is critical to sample negatives that reflect the same degree distribution or node popularity to avoid trivial classification. Often, negatives are sampled uniformly at random or according to a "hard negative" heuristic (e.g., nodes with high degree).
- **Validation Split:** Optionally, partition E_{train} further into training and validation subsets to tune hyperparameters. Care must be taken to ensure validation negatives are drawn from the same distribution as test negatives.

Metrics.

- **Area Under ROC Curve (AUC):** Measures the probability that a randomly chosen positive edge is scored higher than a randomly chosen negative. Less sensitive to class imbalance but may not reflect ranking quality at top-k.
- **Average Precision (AP) / Area Under Precision-Recall Curve:** Focuses on precision over recall; more informative when the positive class is rare (common in sparse graphs).
- **Mean Reciprocal Rank (MRR):** For each positive edge (u, v), rank all candidate negative nodes v' (or both head/tail replacements) by descending predicted score, then compute
$$\text{MRR} = \frac{1}{|E_{\text{test}}|} \sum_{(u,v) \in E_{\text{test}}} \frac{1}{\text{rank}(u, v)}. \tag{2.23}$$
Low computational cost when using a small negative sample per test edge, but can be expensive if ranking over all possible nodes.
- **Hits@K (Top-K Accuracy):** Fraction of positive edges whose true partner appears in the top K of the ranked candidate list. Common choices are $K = 1$, 3, 10. In recommendation, Hits@K aligns with "Does the item appear among the top K suggestions?"
- **Precision@K and Recall@K:** Compute precision and recall over the top-K predictions aggregated over all source nodes. Provides a direct measure of "useful" recommendations in the top slots.

When reporting results, it is important to clarify (1) how negatives were sampled, (2) whether ranking is performed over the entire node set or a sampled subset, and (3) whether metrics are averaged per-node or globally.

Node Classification

Definition. Node classification assigns each node u a label y_u (e.g., user segment, item category). In recommender systems, this can be used to predict a user's demographic cluster or an item's latent category.

Data Splitting.

- **Stratified Sampling:** To preserve label distribution, randomly sample a proportion (e.g., 70%) of labeled nodes for training, 10–15% for validation, and 15–20% for test.
- **Inductive vs. Transductive Setting:**
  - *Transductive:* All nodes (labeled and unlabeled) appear in the graph during training; only labels for a subset are revealed.
  - *Inductive:* A subset of nodes (and their edges) are held out entirely during training. The model must generalize to unseen nodes and their induced subgraph. Inductive evaluation better reflects cold-start scenarios (e.g., a new item with known attributes but no interactions).

Metrics.

- **Accuracy:** Proportion of correctly predicted labels over all test nodes. Appropriate when classes are balanced.
- **Macro/Micro F1-Score:**
  - *Macro-F1* computes F1 per class and averages equally, giving low-frequency classes equal weight.
  - *Micro-F1* computes global counts of true positives, false positives, false negatives across all classes. Appropriate when class imbalance is moderate.
- **ROC AUC (per class or aggregated):** For binary or multi-label classification, one can compute an ROC curve per label.

Graph methods often outperform feature-only classifiers by propagating label information across edges (homophily assumption). However, when node labels are only loosely correlated with topology (heterophily), propagation can harm performance. In such cases, more expressive GNN layers (e.g., higher-order neighborhoods or attention mechanisms) may be required.

2.6.2 Temporal Link Forecasting

In dynamic recommendation contexts, the goal is to forecast future edges (interactions) based on evolving user behavior and item properties. Unlike static link prediction—where edges are assumed independent and identically distributed—temporal forecasting must respect the chronological order of events.

Temporal Hold-Out Strategies

Sliding-Window Evaluation:

1. Choose a fixed time window $[T_{\text{start}}, T_{\text{end}})$ as the training period; construct graph G_{train} using all interactions within this interval.

2. Define a subsequent window $[T_{\text{end}}, T_{\text{end}} + \Delta T)$ as the test period; interactions that occur within this interval are positive test edges.

3. Negative edges are sampled from node pairs that did not interact during $[T_{\text{start}}, T_{\text{end}} + \Delta T)$.

4. Optionally, slide the windows forward by ΔT and repeat to compute performance over multiple time frames, yielding a sequence of metric values that reveal temporal stability or drift.

Incremental Evaluation: Unlike the sliding window, incremental strategies train on $[T_{\text{start}}, t)$ and test on a small next interval $[t, t + \delta)$, then update the training graph to include edges in $[t, t + \delta)$, and iterate. This mimics an online learning scenario and measures performance "in the wild."

Temporal Negative Sampling. Care must be taken not to sample negatives from the future. For each snapshot or time interval, only consider node pairs that have not yet co-occurred up to the current time. Sampling negatives from the entire future graph can artificially inflate performance (data leakage).

Metrics.

- **Time-Aware AUC / AP:** Compute AUC or average precision separately for each test interval; aggregate over time (e.g., average over sliding windows).

- **Temporal MRR / Hits@K:** For each edge (u, v) appearing at time t, rank candidates v' that were not previously connected to u at t by predicted score using the model trained on $[T_{\text{start}}, t)$. Compute per-time-step Hits@K.

- **Time-to-Interaction Error:** For certain applications (e.g., predicting when a user will next purchase an item), one may treat forecasting as a regression problem (predicting timestamp). Metrics such as mean absolute error (MAE) or root mean square error (RMSE) over time predictions become relevant.

Temporal forecasting evaluation inherently suffers from concept drift: user preferences or item relevances can shift over time. As a result, performance metrics often degrade when predicting far into the future. Reporting metric decay curves (metric vs. forecast horizon) provides insight into model robustness.

2.6.3 Influence of Graph Sparsity and Dynamic Updates on Metrics

Real-world recommender graphs are often extremely sparse (e.g., each user interacts with only a tiny fraction of items) and evolve continuously (new users, items, and interactions). These characteristics pose challenges to fair benchmarking and reliable interpretation of evaluation results.[2]

Effects of Sparsity

Class Imbalance: In a graph with $|V|$ nodes and $|E| \ll |V|^2$ edges, the ratio of positive to negative potential edges is overwhelmingly skewed. Consequently, metrics such as accuracy become nearly always high if one naively predicts "no edge" for all pairs. It is crucial to evaluate using ranking metrics (Hits@K, MRR) or precision-recall curves rather than binary accuracy.

Negative Sampling Bias: Random sampling of negatives uniformly over all node pairs can under-represent "hard negatives" (e.g., nodes sharing neighbors). A model might perform well against easy negatives but poorly in realistic scenarios. To counteract this, strategies include:

- **Degree-Aware Sampling:** Sample negative nodes with probability proportional to their degree raised to some power (e.g., $d^{0.75}$), ensuring that popular nodes appear more often as negatives.
- **Distance-Constrained Sampling:** Sample nodes within a certain radius (e.g., two hops) of the source node; these are more likely to be plausible connections.

Evaluation Variance: In very sparse graphs, the absolute number of test edges can be small. Small test sets produce high variance in metrics—reporting standard deviation or confidence intervals (via bootstrap sampling) is recommended to convey reliability.

Impact of Dynamic Updates

Non-Stationarity: As new edges and nodes arrive, the underlying distribution of interactions shifts. An embedding or GNN trained on a static snapshot may become stale quickly. Consequently, one must decide (and clearly state) how often embeddings are retrained or updated—continuously, periodically, or never during the evaluation period.

Temporal Leakage: If node features (e.g., aggregated counts, latest timestamp) are computed over the full dataset (including future interactions) and then used for training, the evaluation is invalid. All node/edge features must be strictly computed using data up to the model's cutoff time.

Metrics Over Time: Instead of reporting a single average metric, it is best practice to report performance as a function of time—e.g., average Hits@K over each weekly interval during the test window. This reveals whether the model degrades or remains stable as the graph evolves.

Benchmarks and Reproducibility

Standard Datasets: Public benchmarks such as CiteSeer, PubMed, MovieLens-KG, Amazon-Bibtex, and Open Academic Graph provide well-curated splits for link prediction and node classification. However, these datasets often differ in size, degree distribution, and annotation quality; care must be taken when comparing results across datasets.

Heterogeneous Graph Suites: Datasets like OGB (Open Graph Benchmark) include both homogeneous and heterogeneous graphs, with standardized splits and evaluation scripts. Using such benchmarks reduces implementation variability.

Reporting Protocols: A reproducible evaluation should include:

1. **Exact Data Split Code:** Seeds for random number generators, sampling ratios, and any filtering criteria.
2. **Negative Sampling Procedure:** Detailed description of how negatives are drawn for training, validation, and testing.
3. **Hyperparameter Search Space:** Ranges and criteria used to select embedding dimension, learning rate, regularization strength, number of GNN layers, etc.
4. **Statistical Significance:** When comparing two models whose average metrics differ by a small margin (e.g., 0.5% in AUC), conduct paired t-tests or bootstrap tests to confirm whether the difference is robust.

Conclusion

We have shown that the languages of triples, attention mechanisms, and message passing provide flexible tools for encoding complex urban relationships. With embeddings that respect proximity, semantics, and inductive generalization—and with graph neural architectures that scale and adapt over time—this chapter lays the groundwork for injecting explicit relational reasoning into the spatio-temporal prediction tasks that follow.

Chapter 3

Spatio-temporal Recommendation systems optimization using knowledge graphs

Introduction

After setting the stage with core machine-learning principles and a deep dive into graph-based representation learning, the discussion now pivots to a single, practical question: does weaving an explicit knowledge graph into a recommender actually sharpen its predictions?

The study that follows translates all the earlier theory into action. First, a city-scale mobility graph is distilled from scraped POIs, administrative polygons and user trajectories, then embedded into compact vectors. Next, three contrasting engines—a plain LSTM sequence model, a temporal-attention network (TGAT), and its memory-augmented cousin (TGN-attn)—are trained in two modes: once with nothing but raw visit sequences, and once with the knowledge-graph signals injected either as static features or as initial memory states. By keeping everything else—data splits, loss functions, negative sampling—strictly identical, any performance gap can be traced to one variable only: the presence or absence of structured knowledge. The upcoming sections describe how the datasets are built, how the embeddings are learned, how each network is configured, and finally how the results reveal when knowledge graphs help, where they hit diminishing returns, and why; see Figure 3.1 for project's pipeline.

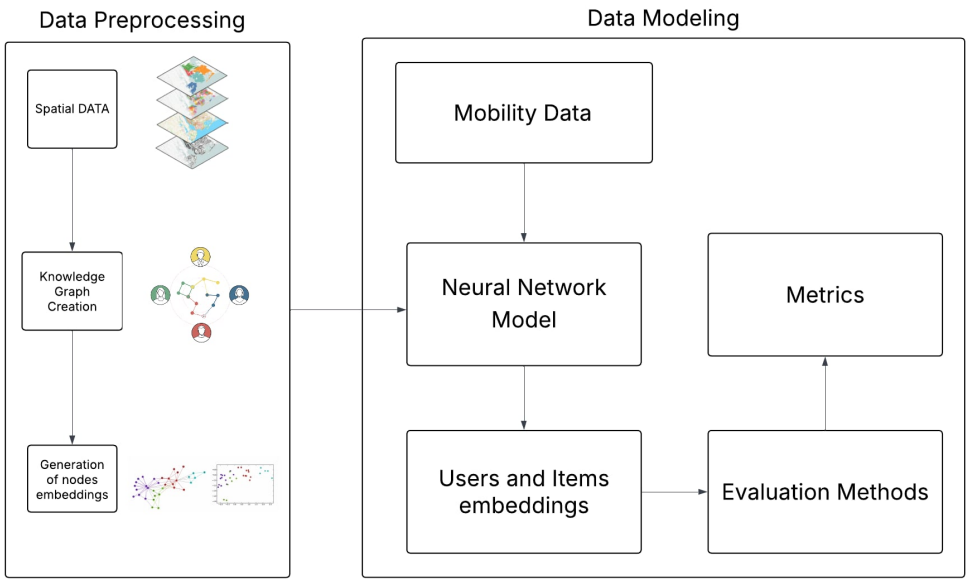


Figure 3.1: Project's pipeline

3.1 Geographical Overview and Administrative Decomposition of Tokyo and San Francisco

3.1.1 Tokyo, Japan

Tokyo Metropolis covers $\approx 2\,194\,\mathrm{km}^2$ and hosts more than 14 million residents. Administratively it functions both as a prefecture and a collection of local governments. (See Figure 3.2)

23 Special Wards (*ku*,).

- Each ward—*e.g.*, Shinjuku, Shibuya, Chiyoda—has its own mayor and council, operating much like an independent city.
- Together they span $\approx 630\,\mathrm{km}^2$ and constitute the dense urban core where most points of interest (POIs) cluster.

Cities, Towns, and Villages outside the Wards ("Tama" region).

- 26 cities (*shi*), five towns (*machi/chō*), and eight villages (*mura*) form the suburban fringe.
- Although less dense, these jurisdictions feed commuters and visitors into the greater Tokyo mobility network.

((a)) Japan's map

((b)) Tokyo's map

Figure 3.2: The images above represent the map of Japan and Tokyo indicating different sub-areas

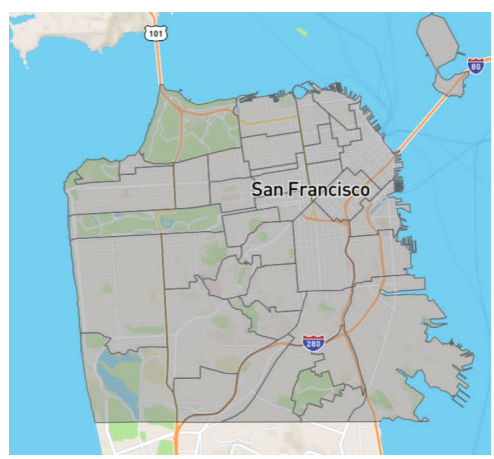
Neighbourhood (*chōme*,) units.

- Within each ward, addresses follow a three–level hierarchy: *chōme* (block)–*ban* (lot)–*gō* (building).

3.1.2 San Francisco, California, USA

San Francisco occupies $\approx 121\,\mathrm{km}^2$ on a Pacific peninsula. The city and county are consolidated, yet two finer partitions are relevant for spatial modelling. (See Figure 3.3)

((a)) California's map



((b)) San Francisco's map

Figure 3.3: The images above represent the map of California and San Francisco

11 Supervisorial Districts.

- Districts 1–11 each elect one supervisor and serve as the principal political subdivisions for budgeting and zoning.
- Example: District 3 covers Russian Hill, Nob Hill, and Chinatown; District 9 encompasses the Mission.

128 Neighbourhoods.

- Maintained by the Planning Department and DataSF as a GeoJSON layer.

3.2 Database Construction for Urban Knowledge Graphs

This section details the construction of spatial and mobility databases that feed the urban knowledge graphs used in the study case. Two cities were selected—**San Francisco (SF)** and **Tokyo (TKY)**.

3.2.1 Spatial Database Development

Data–acquisition strategy. Official geographic data were preferred wherever available; gaps in the Tokyo administrative hierarchy required manual compilation.

- **San Francisco** — full neighbourhood polygons released by the SF Planning Department.
- **Tokyo sources**
  - *Wards* (*ku*): polygons from the Tokyo Metropolitan Government.
  - *Neighbourhoods* (*chōme*): no public polygons; representative coordinates scraped from ward websites and geocoded.
  - *POIs*: OpenStreetMap extracts plus Gowalla check-ins.

3.2.2 Hierarchical Structure Design

Both cities share an identical schema (Table 3.1) to ease cross-city queries; Tokyo's neighbourhood assignment falls back to a coordinate rule due to missing polygons.

3.2.3 Handling Tokyo's Neighbourhood Challenges

Neighbourhood–district assignment.

a) *Coordinate compilation* — scraped or geocoded one representative (`lat`, `lon`) per neighbourhood.

b) *Containment check* — neighbourhood assigned to a ward if its point lies inside the ward polygon.

Borough / District table	`district_id`	Unique ID (e.g. `TKY_12` for Shinjuku)
	`name`	Official administrative name
	`geometry`	Polygon in GeoJSON
Neighbourhood / Area table	`neigh_id`	Unique ID (e.g. `TKY_12_05` Kabukichō)
	`district_id`	FK to borough
	`name`	Local area name
	`lat, lon`	Representative coords (manual) in TK's case, else Polygon
POI table	`poi_id`	Location identifier
	`lat, lon`	WGS-84 coordinates
	`category`	Normalised type (e.g. *restaurant*)
	`district_id`	Parent borough (spatial join)
	`neigh_id`	Closest neighbourhood (distance match)

Table 3.1: Relational schema for both cities

POI–neighbourhood matching (when polygons absent).

a) Compute Euclidean distance from POI to every neighbourhood point.

b) Assign POI to neighbourhood with smallest distance.

3.2.4 Mobility Database Integration

The Gowalla check-in log was enriched with spatial hierarchy:

1. **Coordinate matching**: each record mapped to → a *district* via polygon containment and → a *neighbourhood* via closest point (Tokyo) or polygon (SF).

2. **Edge cases**: check-ins outside either metro area were dropped.

Mobility table schema	
`user_id`	Anonymous visitor ID
`timestamp`	Check-in time (UTC)
`location_id`	Original POI reference
`lat, lon`	Verified coordinates
`category`	Standardised POI type
`district_id`	Matched borough
`neigh_id`	Matched neighbourhood (distance-based in TKY)

Table 3.2: Final mobility data after enrichment

3.2.5 Data-Quality Assurance

- **Tokyo neighbourhoods** — verified that every representative coordinate lies within its assigned ward polygon.
- **POI–neighbour links** — sampled POIs and visually inspected neighborhood assignments on web maps.

Comparison between Tokyo's and SF's data

Comparison shows a larger enrichment of Tokyo's data, which is logical since it covers a larger surface. See Table 3.3

Table 3.3: Basic statistics of the Tokyo and San Francisco UrbanKG datasets

	Tokyo	**San Francisco**
Surface (km^2)	2 194	121
Unique users	1 118	125
Unique POIs	12 686	2 665
Check-ins data	111 736	11 977
Unique categories	209	N/A
Districts	62	11
Neighborhoods	1 583	128

3.3 Knowledge Graph Construction Process

In the following we are going to detail the pipeline that converts raw GIS layers and POI logs into the *UrbanKG* used throughout the experiments, see Figure 3.4.

3.3.1 Data Preparation

Three base layers are loaded and harmonised:

1. **Borough data** — district polygons with unique districtId.
2. **Area data** — neighbourhood polygons with `OBJECTID` and zone names.
3. **POI data** — points of interest with WGS-84 coordinates, category, and visit logs.

Pre-processing steps.

- Re-projected all layers to WGS-84 (EPSG:4326).
- Removed POIs mapped to sentinel ID `999` (no valid borough/area).

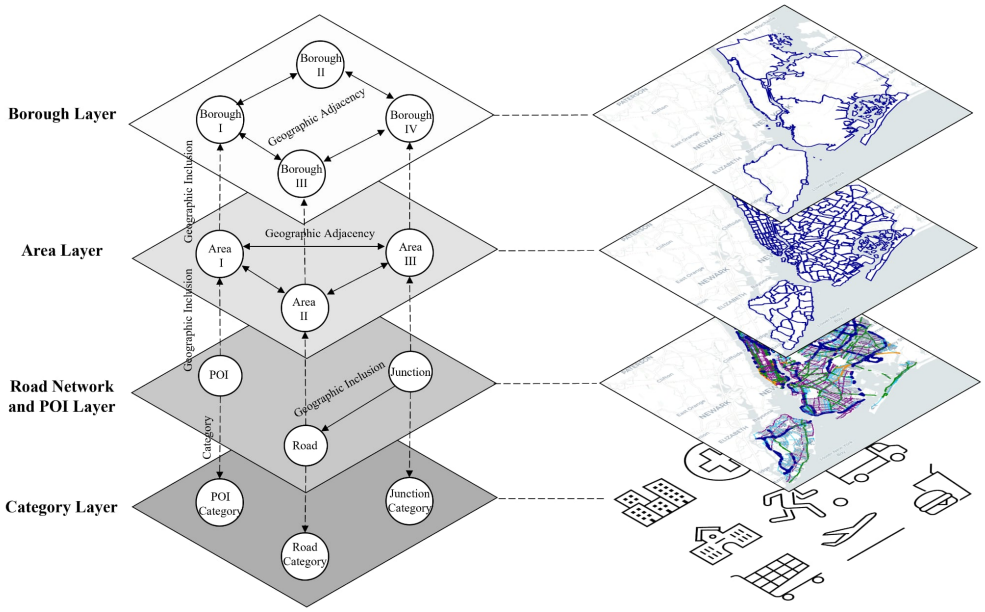


Figure 3.4: Urban Knowledge Graph construction

3.3.2 Spatial Relationship Extraction

Six binary predicates were derived:

1) **Borough Nearby Borough (BNB)**

- Polygon `touches()` test on ward boundaries.
- Example: `Borough/3 BNB Borough/1`.

2) **Area Nearby Area (ANA)**

- Adjacency of neighbourhood polygons.
- Example: `Area/25 ANA Area/26`.

3) **POI Locates at Area (PLA)**

- Point-in-polygon containment.
- Example: `POI/98732 PLA Area/42`.

4) **POI Belongs to Borough (PBB)**

- Hierarchical containment (POI → borough).
- Example: `POI/98732 PBB Borough/1`.

5) **POI Has POI Category (PHPC)**

- Category label attached to POI.
- Example: `POI/98732 PHPC PC/museum`.

6) **Area Locates at Borough (ALB)**

- `within()/intersects()` between area and ward polygons.
- Example: `Area/42 ALB Borough/1`.

3.3.3 Knowledge Graph Assembly

All triples adopt the *head–relation–tail* pattern. Entity prefixes guarantee uniqueness:

`Borough/  Area/  POI/  PC/`

3.3.4 Final Output

The assembled KG is stored as a plain-text file `UrbanKG_NYC.txt`, where each line is a triple, e.g.

```
Borough/3 BNB Borough/1
Area/25 ANA Area/26
POI/98732 PLA Area/42
POI/98732 PBB Borough/1
POI/98732 PHPC PC/museum
Area/42 ALB Borough/1
```

Key features.

1. **Multi-layer hierarchy**: borough → area → POI.
2. **Dual relations**: containment and adjacency captured.
3. **Category integration**: physical nodes linked to semantic types.
4. **Human-readable**: simple parsing while preserving spatial meaning.

This structured KG underpins embedding generation and recommendation tasks by encoding both geographic and categorical relationships among urban entities.

3.4 Knowledge Graph Embedding Generation Process

This appendix documents the pipeline that maps the urban knowledge-graph triples of Tokyo and San Francisco into continuous vector spaces.

3.4.1 Data Preparation

- **Entity / relation mapping** — created dictionaries `entity2id.txt` and `relation2id.txt` that translate prefixed labels into integer IDs (`Borough/1`→ 0, `PLA`→ 2).
- **Dataset split** — 70:15:15 into train/val/test while preserving relation distribution; verified no entity leakage across splits.

3.4.2 Model Configuration

We adopted **TransE** and **GIE** with the hyper-parameters in Table 3.4.

Parameter	**Value**	**Purpose**
Embedding dimension d	32	Latent-space size
Batch size	4 120	GPU–memory fit
Learning rate	10^{-3}	Adam default
Negatives / positive	50	Contrastive learning strength
Training epochs	200 (patience = 10)	Early stopping

Table 3.4: TransE hyper-parameter choices

3.4.3 Training Procedure

Negative sampling. For each positive triple (h, r, t) we drew 50 corrupted triples by replacing either head or tail:

$$(h', r, t), \quad \text{or} \quad (h, r, t').$$

Objective. Margin-ranking loss with $\gamma = 0$:

$$\mathcal{L} = \max\Big(0,\ f_\theta(h, r, t) - f_\theta(h', r, t')\Big), \qquad f_\theta(h, r, t) = \Big\|\mathbf{h} + \mathbf{r} - \mathbf{t}\Big\|_2.$$

Validation. Mean reciprocal rank (MRR) computed every three epochs; training halted after 10 stagnant checks.

3.4.4 Output Embeddings

- **Entity vectors** 32-D for all boroughs, areas and POIs (e.g. `POI/98732`→[0.05, 0.61,...]).
- **Relation vectors** 32-D semantics for every predicate (`BNB`→[0.76, –0.12,...]).
- **Visualisation** 2-D t-SNE: TSNE(n_c=2)$\Big($entity_embeddings$\Big)$.

3.4.5 Results of Knowledge-Graph Embedding Generation

We trained two complementary embedding algorithms—TransE and GIE—on the UrbanKGs for Tokyo and San Francisco, each producing 32-dimensional vectors for every POI entity. Below we summarise both quantitative link-prediction performance and qualitative clustering behavior observed after projecting these embeddings into two dimensions via t-SNE.

Quantitative Link-Prediction Performance For each city, link prediction was evaluated on a held-out test set of triples, measuring Mean Rank (MR), Mean Reciprocal Rank (MRR), and Hits@1/3/10. Recall that a lower MR and higher MRR/Hits@* indicate better relational structure capture.

Tokyo UrbanKG TransE

- Test MR = 25.04
- Test MRR = 0.909
- Hits@1 = 0.884 | Hits@3 = 0.927 | Hits@10 = 0.952

GIE

- Test MR = 75.01
- Test MRR = 0.842
- Hits@1 = 0.809 | Hits@3 = 0.860 | Hits@10 = 0.902

Here, TransE significantly outperforms GIE on Tokyo in both rank and top-k metrics. Its low MR indicates that TransE can almost always place the correct tail within the top few candidates, reflecting the relative simplicity of Tokyo's geographical-hierarchy triples (e.g., "POI → Area → Borough").

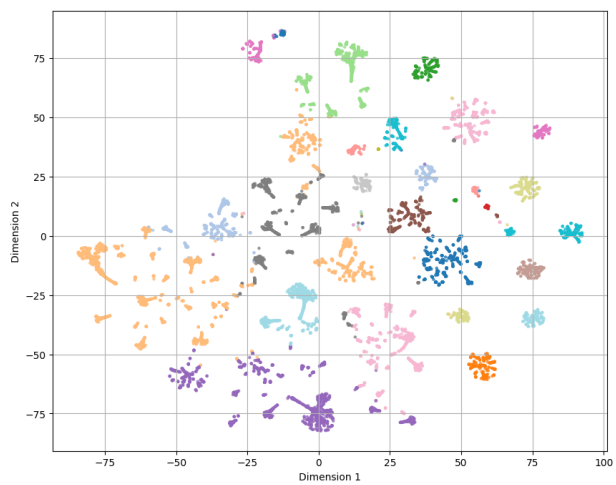
San Francisco UrbanKG TransE

- Test MR = 7.51
- Test MRR = 0.898
- Hits@1 = 0.866 | Hits@3 = 0.918 | Hits@10 = 0.954

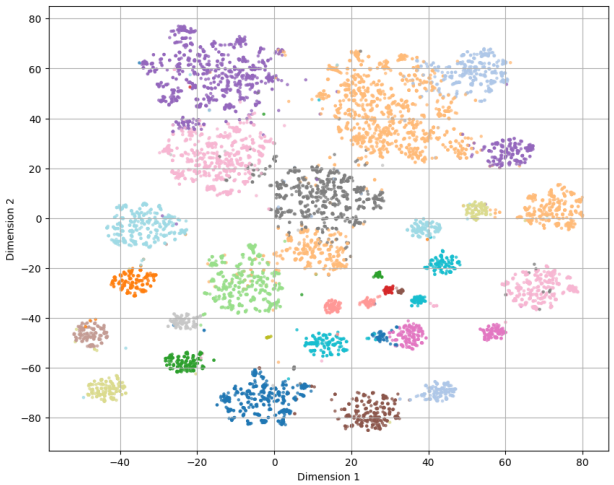
GIE

- Test MR = 16.85
- Test MRR = 0.876
- Hits@1 = 0.846 | Hits@3 = 0.892 | Hits@10 = 0.940

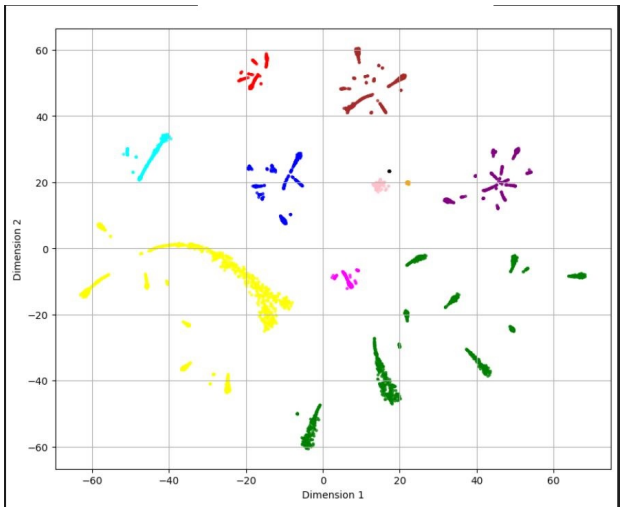
Likewise, TransE outperforms GIE on SF, with an MR nearly half that of GIE. The slightly smaller gaps between TransE and GIE here (compared to Tokyo) suggest that San Francisco's KG—having fewer, larger districts and neighborhoods—requires more nuanced relational encoding, where GIE's inductive neural encoder closes part of the gap. In both cities, however, TransE delivers the strongest single-vector clustering of district–neighborhood relationships.



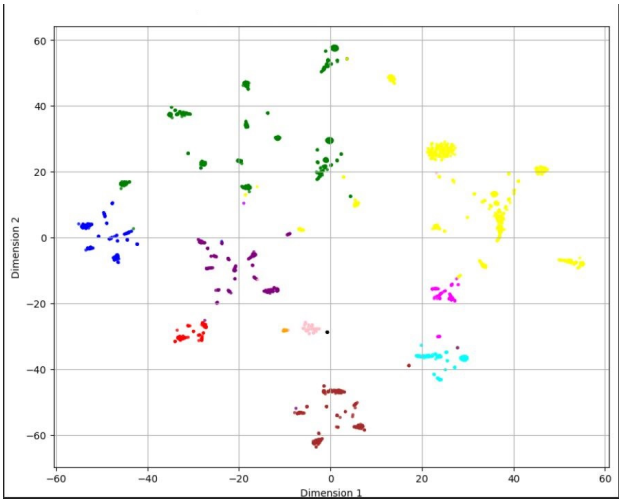
((a)) Tokyo – TransE



((b)) Tokyo – GIE



((c)) San Francisco – TransE



((d)) San Francisco – GIE

Figure 3.5: 2-D t-SNE projections of 32-D POI embeddings, coloured by district. TransE consistently forms tighter, more separable clusters than GIE, especially in Tokyo, which aligns with its lower MR and higher Hits@k.

Qualitative Clustering via t-SNE After training, we collected the 32-D POI embeddings and ran t-SNE (perplexity = 30, learning rate = 200) to project them into 2-D for visualization. We then colored each projected point by its ground-truth district (borough) membership. Across all four models, distinct district-based clusters emerged:

3.5 LSTM-Based Next-POI Predictor: Data Pipeline and Network Layout

This section documents the baseline used in the study to benchmark graph-aware models.

3.5.1 Temporal Batches (*t*-batches)

In the JODIE framework a ***t*-batch** is a set of interaction events that

a) occur within the same contiguous time window Δt, and

b) share *no* users or items with any other t-batch that is processed in parallel.

Construction rule.

1. *Temporal coherence* Every event in the batch must satisfy

$$t_i \;\leq\; t \;<\; t_i + \Delta t,$$

   preserving the chronological order required by recurrent updates.

2. *Node exclusivity* If a user or item appears in one t-batch, it is barred from all other batches handled in the *same* optimisation step. This guarantees that each embedding is updated exactly once per mini-batch, preventing race conditions in the temporal RNN.

These constraints allow JODIE to stream through millions of time-stamped edges while still parallelising across disjoint graph subsets.

Why we adopt *t*-batches To compare different models against JODIE under *identical* training conditions, we generate t-batches using the same Δt and exclusivity rule even though our own models do not strictly require them. This ensures

- each method sees the *same* sequence of events per optimisation step,
- accuracy differences reflect *model architecture* rather than loader bias, and
- runtime measurements remain directly comparable.

Outside of this benchmarking context, t-batches are not used elsewhere in the pipeline.

Run-Time Hyper-Parameters

- ***t*-batch length** $\in \{6\,\mathrm{h}, 12\,\mathrm{h}, 24\,\mathrm{h}\}$
- **Location feature mode**
  (a) `ohe` — one-hot only,
  (b) `emb` — 32-D TransE / GIE vector only,
  (c) `ohe+emb` — concatenation of (a) and (b).

3.5.2 Pipeline Overview

1. **Vector dictionaries**

   - Each `user_id` $\mapsto$ one-hot vector $\mathbb{R}^{|U|}$.
   - Each `location_id` $\mapsto \begin{cases} \text{one-hot} & \texttt{ohe} \\ \text{32-D embedding} & \texttt{emb} \\ \text{concat(one-hot, 32-D)} & \texttt{ohe+emb} \end{cases}$

2. **Event tensors and split**
   `create_batch` groups check-ins into disjoint t-batches; each event stores $\Big[user, curr_loc, t, \Delta_{\text{user}}, \Delta_{\text{loc}},$
   Chronological streaming yields a 70 %/15 %/15 % split (train/val/test).

3. **Input & target**

$$\mathbf{x} = \Big[\text{ user-one-hot} \parallel \text{prev-loc-vec}\Big], \quad \mathbf{y} = \text{current-loc-vec}.$$

$$\text{Input dimension} \begin{cases} |U| + |L| & \texttt{ohe} \\ |U| + 32 & \texttt{emb} \\ |U| + |L| + 32 & \texttt{ohe+emb} \end{cases}.$$

4. **Network architecture**

Block	**Spec**
LSTM	1 layer, 128 hidden units
FC head	Linear$(128, D)$ where $D = \dim(\mathbf{x})$

Table 3.5: Baseline network layout

5. **Training objective**

   - Loss: mean-squared error $\mathcal{L} = \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2$.

- Optimiser: Adam, $\eta = 10^{-3}$.
- Mini-batch: all events in one t-batch (gradients accumulated).
- Early stopping: stop if validation MSE stagnates for 10 checks (≈30 epochs max).

6. **Inference and evaluation**

   a) Save weights to `lstm_model_<feature>_<window>.pth`.
   b) At test time: network emits $\hat{\mathbf{v}} \in \mathbb{R}^D$.
   c) Rank all location vectors by Euclidean distance to $\hat{\mathbf{v}}$; compute Hits@k.

3.6 TGAT Model Architecture

TGAT model is a variant of the TGN framework, with an exclusion of all memory features, which leaves the architecture with just an embedding layer of graph-attention. See Figure 3.6.

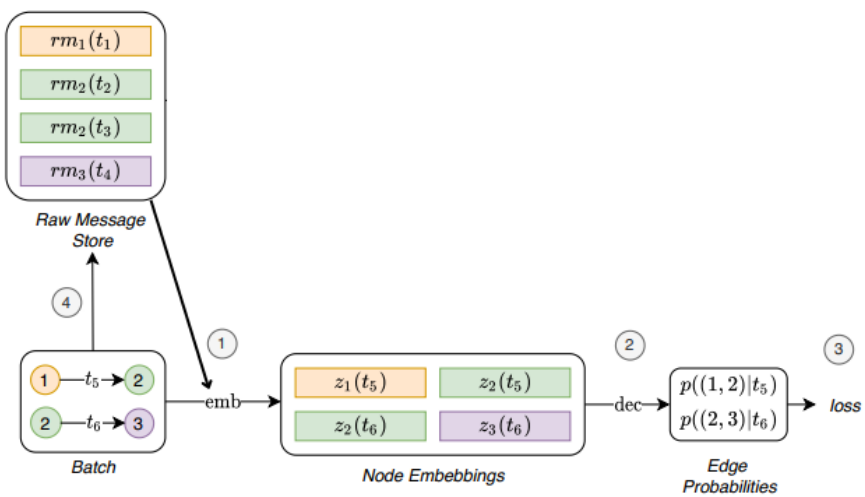


Figure 3.6: TGAT model architecture

3.6.1 Core Components

Temporal embedding layer. Timestamps t are mapped to a continuous feature vector via *Time2Vec*:

$$\boldsymbol{\phi}(t) = \Big[\cos\big(\omega_1 t + \varphi_1\big), \ldots, \cos\big(\omega_d t + \varphi_d\big)\Big],$$

where ω_k and φ_k are learnable parameters. The harmonic basis captures periodic temporal patterns.

Graph-attention mechanism. For every layer ℓ the node representation at query time t is

$$\mathbf{h}_i^{(\ell)}(t) = \mathrm{MLP}^{(\ell)}\Big(\mathbf{h}_i^{(\ell-1)}(t) \parallel \tilde{\mathbf{h}}_i^{(\ell)}(t)\Big), \qquad \tilde{\mathbf{h}}_i^{(\ell)}(t) = \mathrm{MultiHeadAttn}\Big(\mathbf{q}^{(\ell)}(t),\, K^{(\ell)}(t),\, V^{(\ell)}(t)\Big),$$

with keys/values built from the temporal neighbourhood:

$$C^{(\ell)}(t) = \Big[\mathbf{h}_j^{(\ell-1)}(t) \parallel \mathbf{e}_{ij}(t_j) \parallel \boldsymbol{\phi}\Big(t - t_j\Big)\Big]_{j \in \mathcal{N}_i(t)}.$$

Neighbourhood sampling. TGAT samples the k most recent interactions ($k = 20$ by default).

3.6.2 Training Protocol

- **Mini-batching**: chronologically ordered batches of 200 edges.
- **Optimiser**: Adam, learning rate 10^{-4}.
- **Negative sampling**: One negative per positive
- **Early stopping**: monitored on validation AP.

3.6.3 Evaluation Metrics

- Edge prediction (transductive/inductive) with Hits@N ($N \in \{1, 5, 10, 20, 30\}$).
- Standard scores: Average Precision (AP), AUC.

3.6.4 Dataset Format

```
user,item,timestamp,state_label,feature_1,...,feature_n
0,0,0.0,0,0.1,0.3,10.7
2,1,6.0,0,0.2,0.4,0.6
```

3.6.5 Default Configuration

```
python train_self_supervised.py \
  --data mobility \
  --prefix TGN_TGAT \
  --n_layer 2 \
  --n_degree 20 \
  --uniform
```

Key hyper-parameters:

- `n_layer` = 2: two graph-attention layers.
- `n_degree` = 20: 20 temporal neighbours sampled.
- `uniform`: disables recency bias for ablation.

3.7 TGN-Attention (TGN-attn) Model Architecture

A *memory–augmented* extension of temporal graph attention, TGN-attn combines per-node recurrent memory with a graph-attention layer, see Figure 3.7.

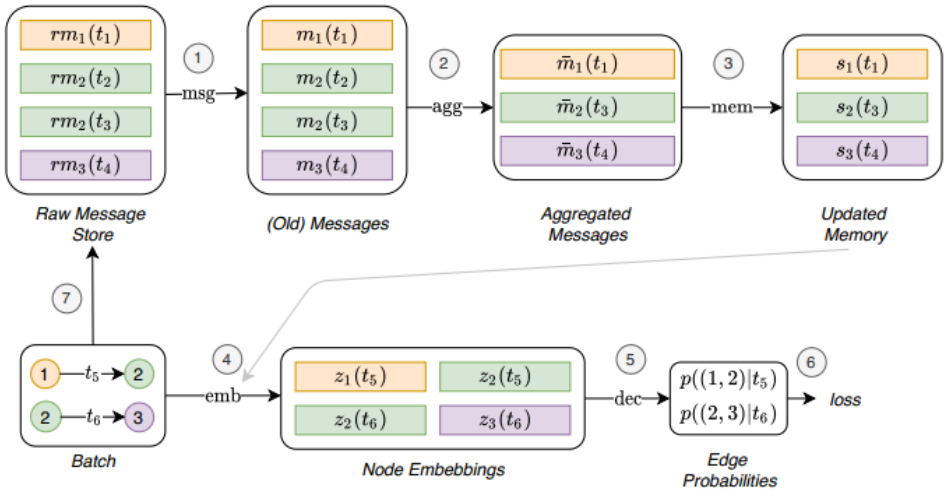


Figure 3.7: Flow of operations of TGN used to train the memory-related modules. Raw Message Store stores the necessary raw information to compute messages, i.e. the input to the message functions, which we call raw messages, for interactions which have been processed by the model in the past. This allows the model to delay the memory update brought by an interaction to later batches. At first, the memory is updated using messages computed from raw messages stored in previous batches (1, 2, 3). The embeddings can then be computed using the just updated memory (grey link) (4). By doing this, the computation of the memory-related modules directly influences the loss (5, 6), and they receive a gradient. Finally, the raw messages for this batch interactions are stored in the raw message store (6) to be used in future batches.

3.7.1 Core Components

The adopted components within our TGN-attn architecture are described in table 3.6.

Block	Description
Temporal memory	Each node v owns a learnable vector $\mathbf{m}_v \in \mathbb{R}^{172}$ (`--memory_dim 172`); updated by a GRU after every batch to accumulate long-term history.
Message construction	For edge (u, v, t) the message $\boldsymbol{\mu}_{uv}(t) = [\mathbf{e}_{uv} \parallel \mathbf{h}_v(t)]$ (identity function).
Message aggregator	`last`: keeps only the most recent message per neighbour inside the batch to preserve temporal order.
Memory updater	$\mathbf{m}_v(t) = \mathrm{GRU}\Big(\mathbf{m}_v(t^-), \overline{\boldsymbol{\mu}}_v(t)\Big)$; run *before* encoding each batch, detached after back-prop to bound gradient depth.
Graph–temporal attention layer	One layer (`--n_layer 1`) with $h = 2$ heads attends over $k = 10$ sampled neighbours (`--n_degree 10`). Queries are $\mathbf{m}_v(t)$; keys/values are neighbours' memories plus $\phi(\Delta t)$.
Time encoding	Same Time2Vec spectrum as TGAT encodes $\Delta t = t - t_{\text{nbr}}$ into $\phi(\Delta t)$.
Edge decoder	One-layer MLP: $\sigma\Big(\phi([\mathbf{h}_s(t) \| \mathbf{h}_d(t) \| \Delta t])\Big) \rightarrow$ interaction probability.

Table 3.6: TGN-attn building blocks

3.7.2 Training Pipeline

Training configuration of hyperparameters is introduced in table 3.7.

Step	Setting
Batching	Chronological batches of 200 edges (`--bs 200`)
Negatives	One negative per positive (`NUM_NEG 1`)
Optimiser	Adam, $\eta = 10^{-4}$
Early stop	Patience = 20 on validation AP
Runs	Five seeds (`--n_runs 5`); metrics averaged

Table 3.7: Training configuration

3.7.3 Evaluation Protocol

- **Transductive**: we predict future links of the nodes observed during training.
- **Inductive**: we predict future links of nodes never observed before.
- **Metrics**: AP, AUC, Hits@K (K=1, 5, 10, 20, 30)—identical to TGAT.

3.7.4 Command-Line Invocation

```
python train_self_supervised.py \
  --data mobility \
  --use_memory \
  --memory_updater gru \
  --embedding_module graph_attention \
  --n_layer 1 \
  --n_degree 10 \
  --aggregator last \
  --message_function identity \
  --prefix tgn-attn-mobility \
  --n_runs 5
```

3.8 Results and findings

City	Metric	LSTM	LSTM_KG	JODIE	JODIE_KG	TGAT	TGAT_KG	TGN_ATTN(1l)	TGN_ATTN(1l)_KG	TGN_ATTN(2l)	TGN_ATTN(2l)_KG
	Acc@1	**9.37**	**9.07**	11.64	**14.23**	5.61T 4.58I	8.22T 6.11I	7.35T 5.48I	9.71T 7.63I	8.66T 7.55I	11.42T 9.34I
TKY	Acc@5	**23.60**	**22.10**	20.33	**30.97**	17.41T 13.98I	22.67T 18.41I	21.69T 16.43I	26.58T 19.26I	32.14T 26.59I	37.76T 30.80I
	Acc@10	**29.01**	**27.45**	33.57	**45.29**	20.59T 17.47I	27.64T 21.36I	27.23T 21.41I	34.36T 28.64I	42.62T 35.46I	49.54T 40.67I
	Acc@20	**34.40**	**32.66**	45.62	**56.31**	25.71T 21.93I	35.26T 30.68I	33.45T 25.79I	44.38T 33.84I	47.95T 39.81I	56.12T 48.93I
	Acc@1	**4.37**	**4.20**	5.72	**7.21**	10.67T 9.59I	12.84T 11.15I	12.39T 10.11I	14.61T 12.44I	13.48T 12.56I	15.93T 14.28I
SF	Acc@5	**10.40**	**9.82**	11.68	**23.49**	24.51T 20.73I	29.19T 25.38I	28.59T 23.64I	33.46T 26.78I	39.05T 33.68I	44.48T 37.86I
	Acc@10	**12.69**	**11.69**	17.34	**30.44**	29.83T 26.57I	36.75T 30.41I	36.21T 30.45I	43.18T 37.55I	52.03T 44.32I	58.36T 49.51I
	Acc@20	**14.61**	**13.90**	28.64	**38.72**	36.58T 32.64I	46.23T 41.48I	44.52T 36.41I	55.14T 44.38I	59.07T 50.58I	67.22T 59.41I

Table 3.8: Full models comparison across cities. "T" and "I" denote transductive and inductive variants.

Table 3.8 presents a comprehensive comparison of prediction accuracy (Acc@k) for each model variant—both with and without integration of Knowledge Graph (KG) embeddings. The purpose of this analysis is not to rank different architectures against each other, but rather to understand the influence of incorporating KG embeddings within each model individually. The following observations arise clearly from these results:

(a) **Impact on the LSTM Model: A Counterintuitive Decline**
Surprisingly, the basic LSTM model demonstrated reduced accuracy when KG embeddings were introduced:

- Tokyo: Acc@10 dropped from 29.01% (LSTM) to 27.45% (LSTM_KG).
- San Francisco: Acc@10 decreased from 12.69% (LSTM) to 11.69% (LSTM_KG).

This unexpected decline suggests that the LSTM's straightforward recurrent architecture lacks the capacity to adequately exploit the richer KG embeddings. Instead of learning meaningful relationships from the additional embeddings, the model treated these high-dimensional vectors as noise, causing confusion and hindering effective generalization. The simplistic structure of the LSTM model likely

limits its ability to integrate and distinguish the useful signals contained within the embeddings from irrelevant fluctuations.

(b) **Superior Performance in Complex Graph-based Architectures**
In contrast, advanced graph-based architectures exhibited clear and consistent gains from the KG embeddings:

- *JODIE* (RNN with co-evolving embeddings):
  - Tokyo: Acc@10 significantly increased from 33.57% to 45.29%.
  - San Francisco: Acc@10 jumped from 17.34% to 30.44%.
- *TGAT* (2-layer attention, no memory):
  - Tokyo: Acc@10 improved from 20.59% to 27.64%.
  - San Francisco: Acc@10 improved from 29.83% to 36.75%.
- *TGN-Attn* (2-layer attention, with memory):
  - Tokyo: Acc@10 improved from 42.62% to 49.54%.
  - San Francisco: Acc@10 improved from 52.03% to 58.36%.

This performance enhancement clearly highlights that sophisticated graph attention and memory mechanisms are better equipped to leverage KG embeddings effectively. These models have the architectural complexity and inductive bias necessary to integrate and aggregate complex signals, allowing the embedded KG vectors to reinforce predictions meaningfully.

(c) **Effect of Attention Layers and Memory Module**
The nuanced comparison between attention depth and memory presence is particularly insightful:

- *More attention layers enhance performance significantly:* Comparing TGN-Attn (1-layer) with TGN-Attn (2-layer), we observe that doubling the number of attention layers dramatically boosts performance:
  - Tokyo: TGN-Attn (1-layer) Acc@10 = 34.36% vs. TGN-Attn (2-layer) Acc@10 = 49.54%.
  - San Francisco: 43.18% vs. 58.36%.

  The additional layer enables the model to aggregate and extract deeper multi-hop relational patterns embedded within the knowledge graph representations.
- *Incorporation of memory further elevates accuracy:* TGN-Attn (2-layer, with memory) outperformed the memory-less TGAT model. The ability to persist historical interaction contexts through memory modules provides additional temporal stability and a richer representation space.

(d) **Synergy between Attention and Memory in TGN**
Comparing TGAT (2-layer attention, no memory) to TGN (1-layer attention, with memory):

- The TGN with a single attention layer and a memory module consistently outperformed the TGAT with two layers of attention but without memory.

- The memory component in TGN appears crucial. It effectively captures historical interactions and temporal dynamics, significantly improving performance.
- The memory module likely provides an additional mechanism for refining embeddings beyond what multiple attention layers alone can achieve.

Summary of Observations:

- Simple architectures (like LSTM) struggle with added complexity, treating KG embeddings as noise due to limited representation and aggregation capabilities.
- Complex architectures (JODIE, TGAT, TGN-Attn) substantially benefit from KG embeddings by effectively filtering and integrating relational information.
- Depth of attention layers significantly boosts predictive performance, highlighting the importance of capturing multi-hop relationships.
- Memory modules complement attention mechanisms, preserving historical context and further refining embedding utilization, particularly beneficial in complex, dynamic prediction tasks.

Thus, the experimental results clearly underscore the necessity of suitable architectural sophistication—particularly attention depth and memory mechanisms—to realize the full potential of knowledge graph embeddings in spatio-temporal prediction tasks.

Conclusion

This chapter empirically evaluated the integration of urban knowledge graphs (KGs) into spatio-temporal recommendation models for Tokyo and San Francisco. We constructed hierarchical KGs, generated effective TransE and GIE embeddings, and verified their spatial semantics through clustering analyses.

The results demonstrated that simpler models (LSTM) struggled to leverage KG embeddings due to limited representational capacity, while advanced architectures (JODIE, TGAT, TGN-attn) achieved notable accuracy improvements. Particularly, deeper attention layers and memory modules emerged as crucial components, successfully exploiting KG relational context. Overall, these findings highlight the significant benefit of incorporating knowledge graphs into sophisticated recommendation architectures.

General Conclusion

This thesis set out to explore how explicit spatial and temporal knowledge—encoded as a hierarchical Urban Knowledge Graph (UrbanKG)—can enhance the accuracy and robustness of next-POI recommendation. Across three chapters, we built a coherent narrative from foundational principles to sophisticated experiments in real cities.

Chapter 1 established the machine-learning bedrock for spatio-temporal recommendation. We reviewed core paradigms (supervised, unsupervised, representation learning), deep-learning building blocks (FCNs, CNNs, RNNs, attention), and rigorous evaluation protocols (MRR, NDCG, Hit@k, temporal hold-outs, statistical significance). This framework clarified the unique challenges of modeling "where" and "when" together—highlighting issues like cold-start, sparsity, and covariate shift that any practical solution must address.

Chapter 2 introduced the graph-centric viewpoint. We defined knowledge graphs, outlined embedding techniques (TransE, GIE, random-walk, neural encoders), and surveyed graph-neural methods—from static GNNs to temporal architectures (TGAT, TGN). We saw how relational context and multi-hop connectivity naturally complement sequence-based approaches, providing a conceptual bridge to richer, more expressive recommenders.

Chapter 3 brought these strands together in a real-world pipeline. We assembled UrbanKGs for Tokyo and San Francisco—linking wards, neighborhoods, and POIs—and learned 32-dimensional embeddings that cleanly clustered by district in t-SNE visualizations. These embeddings were then injected (via concatenation) into four predictive architectures: a baseline LSTM, co-evolving embeddings in JODIE, attention-only TGAT, and memory-augmented TGN-Attn. By holding data splits, negative sampling, and metrics constant, we isolated the effect of KG signals. We discovered that:

- Simple LSTMs lack the capacity to filter noisy, high-dimensional KG vectors, leading to performance degradation.
- Recurrent (JODIE) and attention-based models (TGAT, TGN-Attn) consistently improved with KG embeddings, underscoring their ability to distill relational knowledge.

- Deeper attention (2 layers) was far more effective than a single layer, capturing multi-hop graph structure.
- Memory modules further amplified gains by preserving long-term interaction context, demonstrating a powerful synergy with attention.

Limitations and challenges

Our exploration also revealed practical constraints. Adding per-node memory and multi-layer attention dramatically increased training time and GPU usage, making extensive hyperparameter sweeps—necessary for fair comparison across models and cities—both slow and resource-intensive. We learned that embedding quality alone is insufficient; the downstream architecture must be carefully matched to the embedding complexity.

Future Directions

To extend this work, we recommend:

- **Efficient Memory Designs:** Develop sparse or conditional memory updates to reduce computation while retaining temporal context.
- **Automated Configuration:** Employ Bayesian or gradient-based hyperparameter optimization to streamline architecture search across new urban domains.
- **Multi-Modal UrbanKGs:** Integrate additional data layers (traffic flows, social media, weather) into the KG to enrich embeddings.

In sum, this thesis demonstrates that knowledge graphs, when paired with suitably expressive temporal-graph architectures, can unlock significant improvements in spatio-temporal recommendation. By uniting symbolic urban structure and deep sequence modeling, we lay the groundwork for next-generation, context-aware mobility services in smart cities.

Bibliography

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. New York: Springer, 2006.

[2] Vincent D. Blondel et al. "Fast Unfolding of Communities in Large Networks." In: *Journal of Statistical Mechanics: Theory and Experiment* (2008), P10008.

[3] Antoine Bordes et al. "Translating Embeddings for Modeling Multi-Relational Data." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2013, pp. 2787–2795.

[4] Zongsheng Cao et al. "Geometry Interaction Knowledge Graph Embeddings." In: *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI-22)*. AAAI Press, 2022, pp. —.

[5] Janez Demšar. "Statistical Comparisons of Classifiers over Multiple Data Sets." In: *Journal of Machine Learning Research* 7 (2006), pp. 1–30.

[6] Reinhard Diestel. *Graph Theory*. 5th ed. Berlin: Springer, 2017.

[7] Marta C. González, César A. Hidalgo, and Albert-László Barabási. "Understanding Individual Human Mobility Patterns." In: *Nature* 453 (2008), pp. 779–782.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge, MA: MIT Press, 2016.

[9] Aditya Grover and Jure Leskovec. "node2vec: Scalable Feature Learning for Networks." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2016, pp. 855–864.

[10] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 1024–1034.

[11] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. 2nd ed. New York: Springer, 2009.

[12] Srijan Kumar, Xikun Zhang, and Jure Leskovec. "Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks." In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '19)*. Anchorage, AK: ACM, 2019, pp. 226–236. DOI: 10.1145/3292500.3330895.

[13] Yves-Alexandre de Montjoye, César A. Hidalgo, and Vincent D. Blondel. "Unique in the Crowd: The Privacy Bounds of Human Mobility." In: *Scientific Reports* 3 (2013), p. 1376.

[14] Kevin P. Murphy. *Probabilistic Machine Learning: An Introduction*. Cambridge, MA: MIT Press, 2022.

[15] Yansong Ning et al. "UUKG: Unified Urban Knowledge Graph Dataset for Urban Spatiotemporal Prediction." In: *Proceedings of the 37th Conference on Neural Information Processing Systems, Datasets and Benchmarks Track (NeurIPS 2023)*. Dataset & Benchmarks Track. New Orleans, LA, 2023. URL: https://openreview.net/forum?id=XXXX.

[16] Emanuele Rossi et al. "Temporal Graph Networks for Deep Learning on Dynamic Graphs." In: *arXiv preprint* arXiv:2006.10637 (2020). version dated 9 Oct 2020.

[17] Ryan A. Rossi and Nesreen K. Ahmed. "A Survey of Temporal Graph Analysis." In: *Journal of Big Data* 7.1 (2020), pp. 1–40.

[18] Chaoming Song et al. "Limits of Predictability in Human Mobility." In: *Science* 327 (2010), pp. 1018–1021.

[19] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958.

[20] Zhiwei Sun et al. "Are We Evaluating Rigorously? Benchmarking Link Prediction in Knowledge Graphs." In: *Transactions on Machine Learning Research* 1.1 (2021), pp. 1–23.

[21] Théo Trouillon et al. "Complex Embeddings for Simple Link Prediction." In: *Proceedings of the 33rd International Conference on Machine Learning (ICML)*. 2016, pp. 2071–2080.

[22] Petar Veličković et al. "Graph Attention Networks." In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2018.

Annex

A. JODIE: Joint Dynamic User–Item Embedding Model

JODIE addresses the challenge of modeling sequential user–item interactions by learning continuous embedding trajectories for both users and items. Its key components and workflow are as follows:

Representation:

- Static embeddings: One-hot vectors encoding long-term, time-invariant properties of users/items.
- Dynamic embeddings: Low-dimensional vectors $u(t), i(t) \in \mathbb{R}^d$ that evolve with each interaction, capturing short-term preferences and item states.

Update Operation (Mutually-Recursive RNNs):

- Two RNNCells (RNN_U for users, RNN_I for items) update embeddings at each interaction (u, i, t, f). See Figure 3.8.
- The user RNN ingests the current item embedding $i(t^-)$ plus interaction features f and time-delta Δ_u; symmetrically, the item RNN uses $u(t^-)$ and Δ_i **?**.
- This co-evolution ensures that each entity's embedding reflects its partner's evolving state.

Projection Operation:

- Learns to forecast a user's embedding at any future time $t + \Delta$.
- Uses a learned temporal attention vector (via Hadamard interaction with Δ) to drift the embedding appropriately: small drift for short Δ, larger for long gaps.
- Enables embedding prediction in constant time by nearest-neighbor search (e.g., LSH), rather than scoring all items.

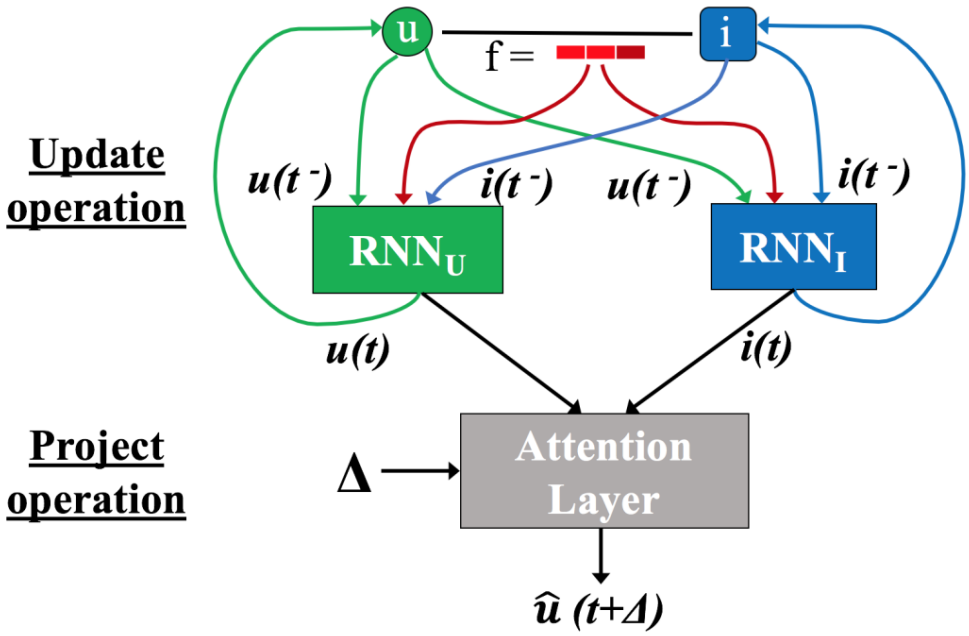


Figure 3.8: JODIE model architecture

t-Batch Training:

- To scale beyond sequential updates, the t-Batch algorithm partitions interactions into independent but time-consistent batches: each user/item appears at most once per batch, and batch indices respect temporal order.
- This permits parallel processing, yielding a $\sim 9\times$ speed-up over sequential baselines.

Learning Objective:

- Trains both update and projection operations to minimize the MSE between predicted item embeddings and their true dynamic embeddings at the next interaction.
- Negative sampling is unnecessary due to direct embedding regression.

Inference & Applications:

- Next-item prediction: Project user embedding to query time, then retrieve closest item embedding.
- State change prediction: Uses projected embeddings to foresee events like user churn or dropout.
- Achieves at least 20% relative improvement in MRR for future interactions and 12% in AUC for state-change tasks over strong baselines, while being $9.2\times$ faster than DeepCoevolve.

Overall, JODIE's coupled RNN updates, learned projection, and batching strategy form a cohesive framework for capturing both co-evolving entity dynamics and temporal drift, delivering scalable, accurate predictions in temporal interaction networks.