# KGAT Book Recommendation System: Attention Mechanism and Implementation

Getachew Getu

May 2025

**Abstract**

This work provides a detailed explanation of the attention mechanism in the Knowledge Graph Attention Network (KGAT) for Book recommendation systems (Task 1) and describes the implementation of a KGAT-based recommendation system using the Book-Crossing dataset, including data processing, model training, and visualization of recommendations (Task 2). The attention mechanism in KGAT prioritizes relevant neighbors in a knowledge graph to enhance recommendation quality, while the implementation demonstrates a practical application with visualizations of recommended items and contributing knowledge graph paths.

# 1 Introduction

## 1.1 Overview

With the rapid development of the internet and the exponential growth of data, it is difficult for users to pick what interests them. In the current world of Artificial Intelligence, recommendation systems find applications in almost every domain ranging from news portals, social media platforms, e-commerce, music, and search engines. Thus, identifying user preferences and suggesting the best products for them is currently an important area of research.

However, most recommendation systems currently don't go beyond modeling user-item interactions. Factorization or collaborative methods are based on assumptions that each interaction is an independent instance encoding side information. This causes an overlook of the relations among instances or items. Thus, to provide more diverse, accurate, and

explainable recommendations, we need to model side information and create a collaborative signal bringing in many more users.

In this work, we bring out Knowledge Graph Attention Network (KGAT) for Book Recommendations that models these high-order connectivity. The KGAT is a graph-based recommendation model that integrates user-item interactions and relational knowledge from a knowledge graph (KG). The attention mechanism in KGAT plays a critical role in determining which neighbors in the graph are more important for generating accurate recommendations. In a KG, nodes represent entities (e.g., users, items, attributes like authors), and edges represent relations (e.g., "rated", "written_by"). The attention mechanism allows KGAT to focus on the most relevant neighbors, filtering out noise and enhancing personalization.

## 1.2 How Attention Works in KGAT

The attention mechanism in KGAT operates during the message-passing phase, where information from neighboring nodes is aggregated to update each node's embedding. Below is a detailed explanation of the process:

- **Graph Structure and Embeddings**
  - Each entity (e.g., user, book, author) is assigned an embedding vector via an embedding layer. Similarly, each relation type (e.g., "rated", "written_by") has an embedding.
  - For an edge in the KG, represented as a triple *(h, r, t)* (head entity, relation, tail entity), the model retrieves the embeddings $e_h$, $e_r$, and $e_t$ for the head, relation, and tail, respectively.
- **Attention Score Computation**
  - The importance of a neighbor (tail entity t) for a head entity h under relation r is quantified by an attention score $\alpha_{h,r,t}$.
  - The head and relation embeddings are concatenated: $e_h \,\|\, e_r$.
  - This concatenated vector is passed through a linear layer followed by a sigmoid activation to compute the attention score:

$$\alpha_{h,r,t} = \sigma(W \cdot [e_h \,\|\, e_r]),$$

where W is a learnable weight matrix, and σ is the sigmoid function, producing a score between 0 and 1

➢ **Weighted Aggregation of Neighbors:**

- The attention score $\alpha_{h,r,t}$ weights the tail entity's embedding et, producing a weighted contribution: $\alpha_{h,r,t} \cdot e_t$.
- For each head entity, the weighted embeddings of all its neighbors are summed to form an aggregated embedding:

$$e_h' = \sum_{(h,r,t)\in N_h} \alpha_{h,r,t} \cdot e_t$$

- This aggregated embedding is transformed via a linear layer and a non-linear activation (e.g., tanh) to update the head entity's embedding:

$$e_h'' = \tanh(W' \cdot e_h'),$$

where W ′ is another learnable weight matrix.

## 1.3 Why Attention Matters

The attention mechanism is crucial for the following reasons:

- **Selective Focus:** Not all neighbors are equally relevant. For a user, books they rated highly or authors they prefer are more informative than low-rated books. Attention assigns higher weights to these neighbors.
- **Relation Awareness:** By incorporating relation embeddings, the model differentiates between relation types (e.g., "rated" vs. "written_by"), prioritizing those most relevant to the recommendation task.
- **Personalization:** Attention weights are learned during training, tailoring the aggregation to each entity's context, thus capturing user-specific or item-specific patterns.
- **Noise Reduction:** In a heterogeneous KG with diverse entities and relations, attention filters out less relevant connections, improving embedding quality.

# 2 Building KGAT Recommendation System

## 2.1 Overview

Task 2 involves implementing a KGAT-based recommendation system using the Book-Crossing dataset, transforming it into a knowledge graph, training a KGAT model, and visualizing recommendations for sample users. The system leverages user-book interac- tions and book-author relationships to recommend books, with visualizations showing recommended items and contributing KG paths.

## 2.2 Dataset collection and preparation

### 2.2.1 Book-Crossing Dataset

The Book-Crossing dataset is chosen for its suitability for a KG-based recommendation system. It consists of three CSV files:

- **BX-Books.csv**: Contains book metadata (ISBN, title, author, year of publication, publisher).
- **BX-Users.csv:** Contains user information (user ID, location, age).
- **BX-Book-Ratings.csv:** Contains user-book ratings (user ID, ISBN, rating from 0 to 10).

### 2.2.2 Preprocessing

- **Books:** The Year-Of-Publication column is converted to numeric, with invalid entries set to NaN. Relevant columns are selected and renamed (e.g., Book-Title to title).
- **Users and Ratings:** Columns are renamed for consistency (e.g., User-ID to user_id, Book-Rating to rating).
- **Filtering:** Users with at least 200 ratings are selected to focus on active users. Books with at least 50 ratings are retained to ensure popularity. Duplicate user-book interactions are removed.
- The resulting dataset (final_rating) contains filtered user-book interactions merged with book metadata.

## 2.3 Knowledge Graph Construction

The dataset is transformed into a knowledge graph with the following components:

- **Entities (Nodes):**

    - Users: Identified by user_id, encoded as user_id_encoded using LabelEncoder.
    - Books: Identified by title, encoded as book_id_encoded.
    - Authors: Identified by author, encoded as author_encoded.
- **Relations (Edges):**
    - Rated: Connects users to books based on rating interactions.
    - Written_by: Connects books to their authors.
- **Implementation:**
    - Edges are stored as triples (source, destination, relation) in kg_edges.
    - The graph is converted to tensors: edge_index (shape [2, num_edges]) for source-destination pairs and edge_type for relation IDs (0 for "rated", 1 for "written_by").

## 2.4 KGAT Model Implementation

### 2.4.1 Model Architecture

The KGAT model is implemented from scratch in PyTorch, extending nn.Module. Key components include:

- **Entity Embeddings:** An embedding layer maps each entity (user, book, author) to a 64-dimensional vector.
- **Relation Embeddings:** Maps each relation type to a 64-dimensional vector.
- **Attention Layer:** A linear layer computes attention scores from concatenated head and relation embeddings.
- **Transformation Layer:** A linear layer transforms aggregated embeddings. Embeddings are initialized using Xavier normal initialization.

### 2.4.2 Attention Implementation in the Model

In the provided KGAT implementation, the attention mechanism is implemented in the forward method of the KGAT class:

- Concatenation: head_rel = torch.cat([head, rel], dim=-1).
- Attention Score: att_scores = torch.sigmoid(self.attention(head_rel)).
- Weighted Aggregation: neighbor_emb = att_scores * tail, followed by manual summation over neighbors.
- Transformation: entity_emb = torch.tanh(self.W(aggr_emb)).

This process ensures that the model prioritizes neighbors that contribute most to accurate recommendations.

### 2.4.3 Forward Pass

The forward method processes the knowledge graph:

- Retrieves head, tail, and relation embeddings for each edge.
- Computes attention scores using concatenated head and relation embeddings, followed by a sigmoid activation.
- Aggregates neighbor embeddings by weighting them with attention scores and summing manually.
- Updates entity embeddings using a linear transformation and tanh activation.

The predict method computes recommendation scores as the dot product of user and item embeddings.

### 2.4.4 Training

- The model is trained for 10 epochs using the Adam optimizer (learning rate 0.001).
- Loss is computed as Mean Squared Error (MSE) between predicted and actual ratings.
- Training data consists of user-book-rating triples, with book IDs offset to align with the entity embedding space.

### 2.4.5 Recommendation Generation

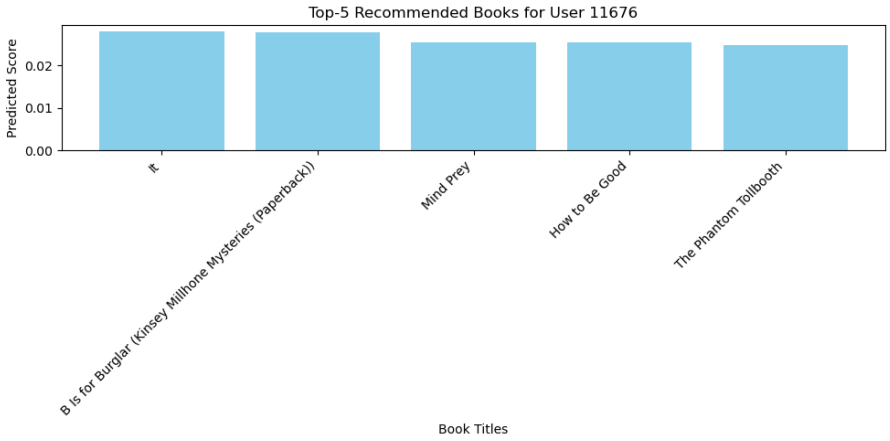
A recommend_books function generates top-5 recommendations for a given user:

- Encodes the user ID and computes scores for all books using the predict method.
- Selects the top-5 books using torch.topk.
- Decodes book IDs to titles using the book_encoder.
- Returns both the recommended book titles and their indices for visualization.

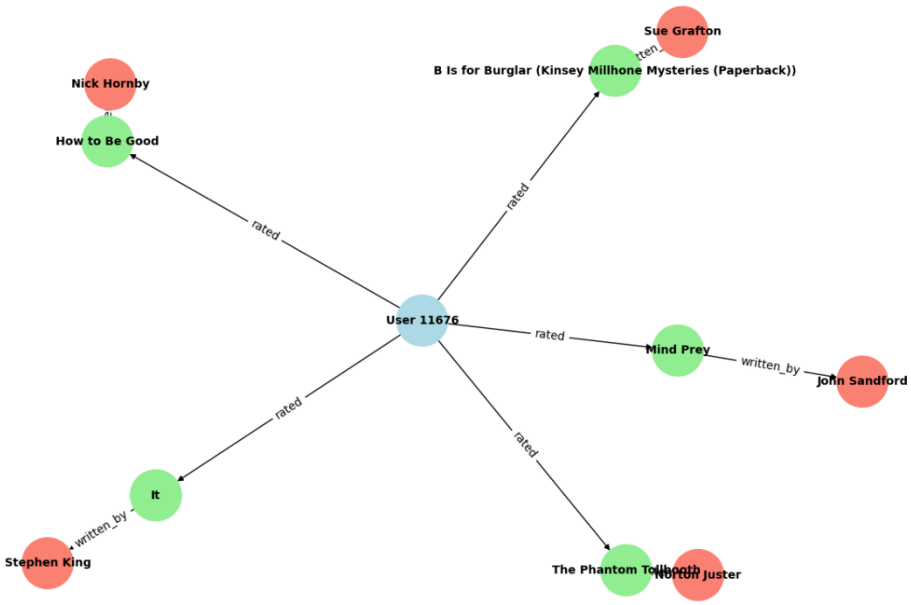
### 2.4.6 Visualizations

Two visualizations are implemented to display recommendations and contributing KG paths:

- **Bar Chart (visualize_recommendations):**
  - Displays the top-5 recommended books and their predicted scores for a sample user.
  - Uses Matplotlib to create a bar chart, with book titles on the x-axis and scores on the y-axis.




- **Network Graph (visualize_kg_subgraph):**
  - Visualizes a subgraph of the KG, including the user, recommended books, and their authors.
  - Uses NetworkX to create a directed graph, with nodes colored by type (user: light blue, book: light green, author: salmon) and edges labeled by relation ("rated", "written_by").




## 2.4.7 Implementation Component used and Details

- Libraries: Pandas, NumPy, PyTorch, Scikit-learn (for LabelEncoder), Matplotlib, and NetworkX.
- Device: The model uses CPU.

- Artifacts: The trained model and encoders are saved using pickle to artifacts/ directory.
- Testing: Recommendations and visualizations are tested for the first active user in the filtered dataset.

# 3 Results and Visualizations

- Recommendations: The system outputs the top-5 book titles for a sample user, printed to the console.
- Bar Chart: Shows the relative predicted scores, indicating the model's confidence in each recommendation.
- Network Graph: Illustrates the KG paths (e.g., user → book → author) that contribute to the recommendations, highlighting the role of relational knowledge.

## 3.1 Limitations and Potential Improvements

- Scalability: Manual aggregation in the forward method is inefficient for large graphs. Using PyTorch Geometric's scatter_sum could improve performance.
- Hyperparameters: Fixed embedding dimension (64) and epochs (10) could be tuned for better performance.
- Dataset Sparsity: The filtered Book-Crossing dataset is sparse. A denser dataset could improve recommendation quality.

# 4 Conclusion

The KGAT model's attention mechanism effectively prioritizes relevant neighbors in a knowledge graph, enhancing recommendation accuracy by focusing on informative entities and relations. The implementation using the Book-Crossing dataset demonstrates a practical KGAT-based recommendation system, with visualizations that clearly convey recommended items and the underlying KG paths. Future improvements could include scalability enhancements, quantitative evaluation, and hyperparameter tuning to further refine the system.