Chapter 1

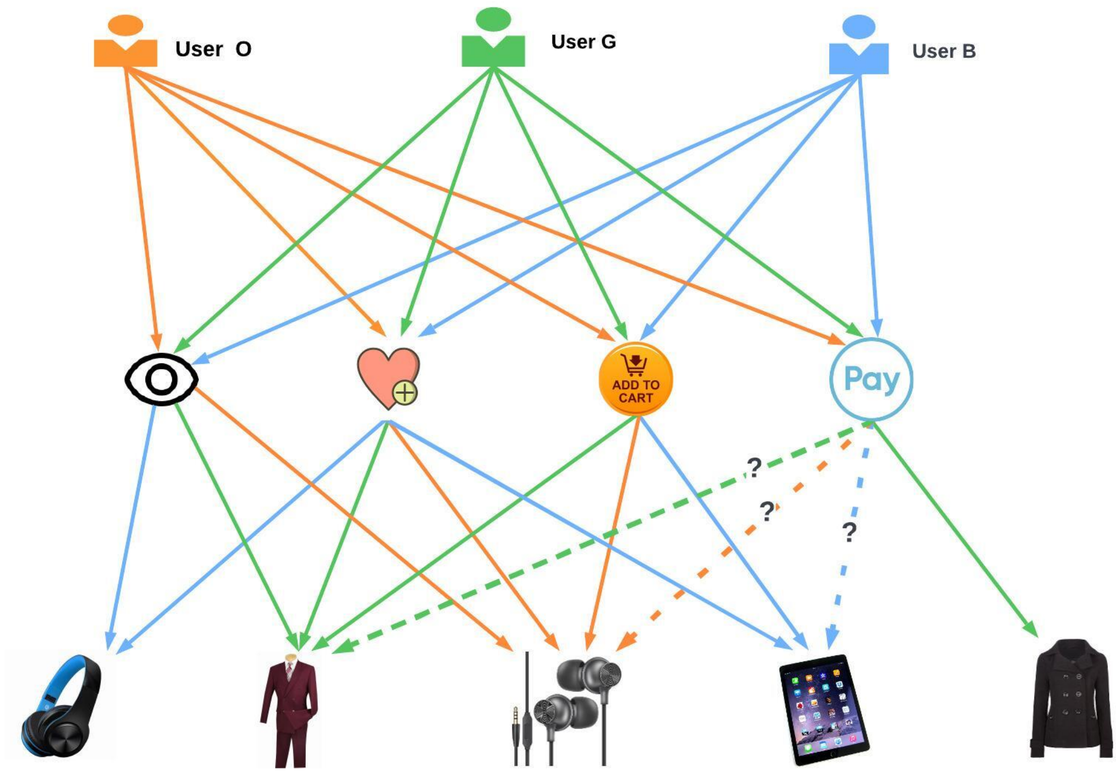

Figure 1.1: An example of a real-world e-commerce recommendation scenario involving multiple user behaviour types. Hyperedges (view, add-to-cart, add-to-favourites, purchase) allow modelling many-to-many relationships in a single structure e.g., behaviour-specific and co-interaction patterns.

1.1 Methodology

1.1.1 Hypergraph Encoder

Given a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the node set $\mathcal{V} = U \cup I$ consists of users U and items I, and the hyperedges $\mathcal{E}$ capture multi-behaviour interactions such as views, cart additions and purchases, the hypergraph encoder produces compact user-item representations through the following steps:

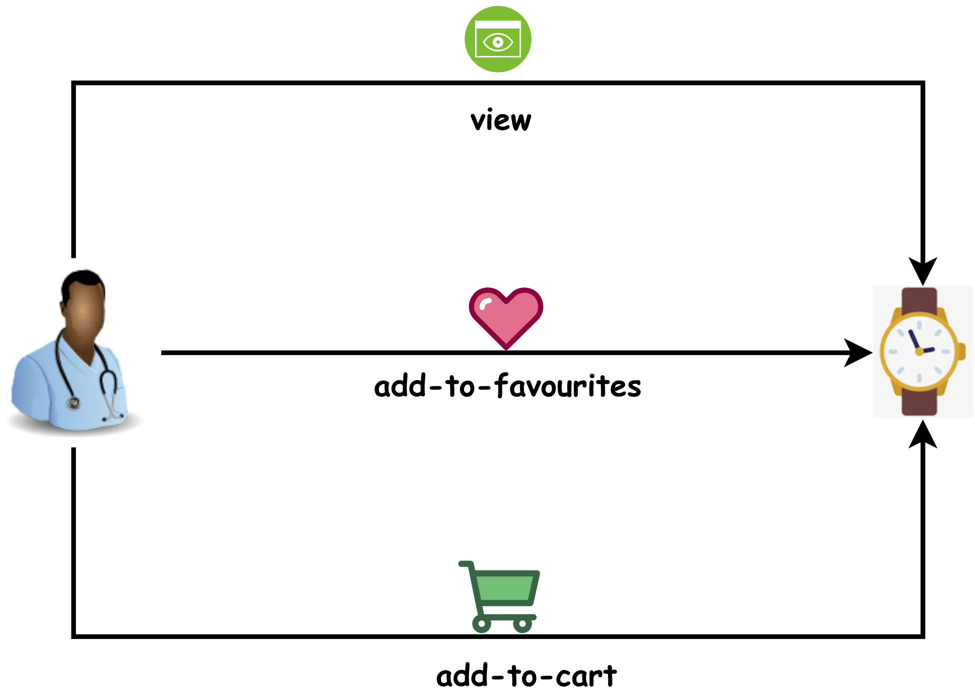

Figure 1.2: Example of a real-world higher-order user-item interaction involving triadic behaviour: *"view"*, *"add-to-cart"* and *"add-to-favourites"*. GNN methods are limited to pairwise interactions and do not effectively model higher-order relations. Hypergraphs can naturally model these muti-behaviour patterns.

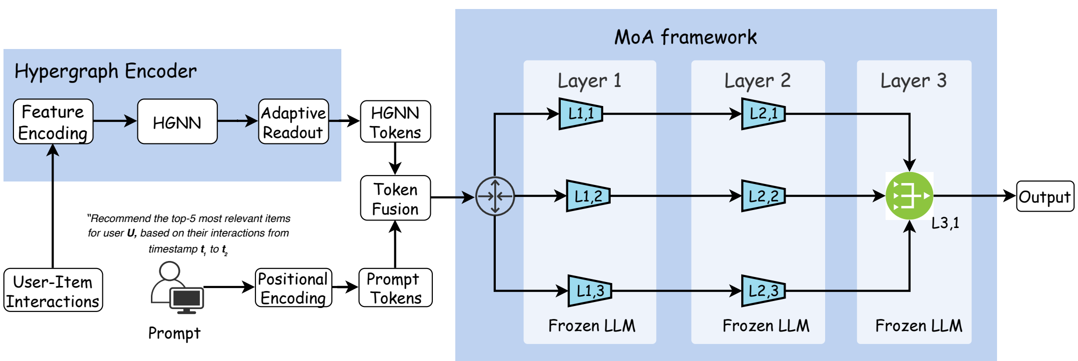

Figure 1.3: Architecture of the HGLMRec model. The prompt input passes through an MoA framework, each containing LLM agents. These agents leverage on user-item interactions captured by the hypergraph encoder . The final MoA layer (L3,1) aggregates information from the intermediate LLM agents.

Feature Initialization. Each node $v \in \mathcal{V}$ is initialized with a learnable embedding vector:

$$\mathbf{h}_v^{(0)} = \begin{cases} \mathbf{E}_u[v] \in \mathbb{R}^d & \text{if } v \in U, \\ \mathbf{E}_i[v] \in \mathbb{R}^d & \text{if } v \in I, \end{cases} \tag{1.1}$$

where $\mathbf{E}_u$ and $\mathbf{E}_i$ are embedding lookup tables for users and items respectively, and d is the embedding dimension.

Hypergraph Convolution. To capture higher-order user-item interactions, we apply two layers of hypergraph convolution. At each layer $l \in \{0, 1\}$, node features are updated by aggregating normalised messages from all hyperedges:

$$\mathbf{h}_v^{(l+1)} = \text{LayerNorm}\left(\sigma\left(\sum_{e \in \mathcal{E}(v)} \frac{1}{|e|} \sum_{u \in e} \mathbf{h}_u^{(l)} \mathbf{W}^{(l)}\right)\right), \tag{1.2}$$

where $\mathcal{E}(v)$ is the set of hyperedges containing node v, $|e|$ is the size of hyperedge e, $\mathbf{W}^{(l)} \in \mathbb{R}^{d \times d}$ is a learnable weight matrix, σ is the ReLU activation function, and *LayerNorm* stabilises training.

Adaptive Readout. To improve the representation learning of user-item interactions from the HGNN module, we apply adaptive readout [1], a function that aggregates the HGNN embeddings [5]. In HGLMRec, readout is applied using attention-weighted grouping (Equation 1.3) that retains expressive hypergraph summaries tailored to the particular interactions for the particular iteraction.

Token Generation. After two convolutional layers, node embeddings $\{\mathbf{h}_v^{(l)}\}_{v \in \mathcal{V}}$ are pooled by leveraging *adaptive readout* to form compact graph tokens:

$$\alpha_v = \frac{\exp\left(\mathbf{a}^\top \tanh\left(\mathbf{W}_a \, \mathbf{h}_v^{(l)}\right)\right)}{\sum_{u \in \mathcal{V}} \exp\left(\mathbf{a}^\top \tanh\left(\mathbf{W}_a \, \mathbf{h}_u^{(l)}\right)\right)}, \quad \mathbf{G} = \text{MLP}\left(\sum_{v \in \mathcal{V}} \alpha_v \, \mathbf{h}_v^{(l)}\right) \tag{1.3}$$

where α_v are attention weights learned via $\mathbf{W}_a$ and $\mathbf{a}$, and the MLP ensures flexible mapping of aggregated features.

1.1.2 Token Fusion

To align graph-based and prompt signals, HGLMRec fuses $\mathbf{G}$ with a tokenised task prompt. We apply token fusion by concatenation [3] to the model that combines structured graph tokens with the recommendation task prompt [4]. Specifically, a prompt such as "*Recommend the top-5 most relevant items for user U, based on their interactions from timestamp t2 to t1*", is tokenised, which converts the text into a sequence of token embeddings $\mathbf{P} \in \mathbb{R}^{m \times d}$, where m is the number of prompt tokens and d is the embedding dimension. These prompt embeddings $\mathbf{P}$ are then concatenated with hypergraph tokens $\mathbf{G} \in \mathbb{R}^{k \times d}$, which summarise user-item interactions learned from the hypergraph encoder. To retain the position information of the tokens in the combined sequence, the positional encoding $\mathbf{P}_{\text{pos}} \in \mathbb{R}^{(k+m) \times d}$ is applied [2]. The fused input tokens are then passed into the downstream MoA agents, which aligns HGNN and the recommendation task representations in a shared embedding space with positional context.

$$\mathbf{x}_1 = \text{Concat}(\mathbf{G}, \mathbf{P}) + \mathbf{P}_{\text{pos}} \tag{1.4}$$

.

1.1.3 Mixture-of-Agents Framework

The Mixture-of-Agents (MoA) module processes the fused input tokens to iteratively refine recommendation predictions. Specifically, the MoA consists of multiple layers, each containing several agents denoted by $A_{i,j}$, where i indexes the layer and j indexes the agent within that layer. At each layer i, the input token representation $x_i \in \mathbb{R}^{m \times d}$ is processed in parallel by all n agents $A_{i,1}, \ldots, A_{i,n}$. Each agent $A_{i,j}(\cdot)$ is a "frozen" LLM that produces refined token embeddings. The outputs of all agents in the layer are combined using a cross-agent attention-based aggregation operator, denoted by $\bigoplus$. This aggregated output is then combined with the initial fused input tokens $\mathbf{x}_1$ through a residual connection, producing the intermediate representation

y_i, which serves as input x_{i+1} for the next layer.

$$y_i = \bigoplus_{j=1}^{n} A_{i,j}(x_i) + \mathbf{x}_1, \quad \text{and} \quad x_{i+1} = y_i. \tag{1.5}$$

The LM agents are "frozen"; only the hypergraph encoder and MLP layers are trained. The final recommendation prediction y_{final} is obtained by applying a trainable MLP followed by a *softmax* function on the output of the agent in Layer 3 is computed as follows:

$$y_{\text{i}} = \text{Softmax}\Big(\text{MLP}(A_{i,j}(x_i))\Big). \tag{1.6}$$

.

1.1.4 Algorithms

- **Hypergraph Encoder (Algorithm 1)**.
- **Prompt-Guided Reasoning (Algorithm 2)**
- **Optimization (Algorithm 3)**

Algorithm 1 Hypergraph Encoder

Require: Hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, user/item sets U, I
Ensure: Graph tokens $\mathbf{G} \in \mathbb{R}^{k \times d}$
Initialize embeddings: $\mathbf{E}_u \in \mathbb{R}^{|U| \times d}$, $\mathbf{E}_i \in \mathbb{R}^{|I| \times d}$
for each node $v \in \mathcal{V}$ **do**
    $\mathbf{h}_v^{(0)} \leftarrow \mathbf{E}_u[v]$ if $v \in U$; else $\mathbf{E}_i[v]$
end for
for $l = 0$ to 1 **do**
    for each node $v \in \mathcal{V}$ **do**
        $\mathbf{z}_v \leftarrow \sum_{e \in \mathcal{E}(v)} \frac{1}{|e|} \sum_{u \in e} \mathbf{h}_u^{(l)} \mathbf{W}^{(l)}$
        $\mathbf{h}_v^{(l+1)} \leftarrow \text{LayerNorm}(\text{ReLU}(\mathbf{z}_v))$
    end for
end for
$\mathbf{G} \leftarrow \text{MLP}(\text{MeanPool}(\{\mathbf{h}_v^{(2)}\}_{v \in \mathcal{V}}))$
return G

Algorithm 2 Prompt-Guided Reasoning

Require: Graph tokens $\mathbf{G}$, prompt T, agents $\{A_{i,j}\}$
Ensure: Predicted scores $\mathbf{y}_{\text{final}} \in \mathbb{R}^{|I|}$
$\mathbf{P} \leftarrow \text{MoA\_Tokenizer}(T)$
$\mathbf{x}_1 \leftarrow \text{Concat}(\mathbf{G}, \mathbf{P}) + \text{PE}()$
for $i = 1$ to 3 **do**
    $n \leftarrow \{3, 3, 1\}[i]$
    for $j = 1$ to n **do**
        $\mathbf{o}_j \leftarrow A_{i,j}(\mathbf{x}_i)$
    end for
    if $i < 3$ **then**
        $\mathbf{y}_i \leftarrow \text{MoA}(\mathbf{x}_i, \{\mathbf{o}_j\}) + \mathbf{x}_1$
        $\mathbf{x}_{i+1} \leftarrow \mathbf{y}_i$
    else
        $\mathbf{y}_{\text{final}} \leftarrow \text{Softmax}(\text{MLP}(\mathbf{o}_1))$
    end if
end for
return $\mathbf{y}_{\text{final}}$

Algorithm 3 Training Procedure

Require: Dataset $\mathcal{D}$, epochs E, batch size B, learning rate η
Ensure: Trained parameters Θ
Initialize AdamW optimizer with linear warmup (500 steps)
for epoch $= 1$ to E **do**
    for each batch $(G, T, A) \in \mathcal{D}$ **do**
        $\mathbf{G} \leftarrow$ Algorithm 1(G)
        $\hat{A} \leftarrow$ Algorithm 2$(\mathbf{G}, T)$
        Compute loss: $\mathcal{L}$
        Update: $\Theta \leftarrow \Theta - \eta \nabla_{\Theta} \mathcal{L}$
    end for
    Log validation metrics
end for

Bibliography

[1] David Buterez et al. “Graph neural networks with adaptive readouts”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 19746–19758.

[2] Florian Grötschla, Jiaqing Xie, and Roger Wattenhofer. “Benchmarking Positional Encodings for GNNs and Graph Transformers”. In: *arXiv preprint arXiv:2411.12732* (2024).

[3] Jia Li et al. “Graph intelligence with large language models and prompt learning”. In: *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2024, pp. 6545–6554.

[4] Bryan Perozzi et al. “Let your graph do the talking: Encoding structured data for llms”. In: *arXiv preprint arXiv:2402.05862* (2024).

[5] Keyulu Xu et al. “How powerful are graph neural networks?” In: *arXiv preprint arXiv:1810.00826* (2018).