
Learning a Multi-Head Value Model for Short-Video Recommendation with GNNs

Haozhan Gao

Stanford Graduate School of Business
hgao24@stanford.edu

Abstract

Recommender systems rely on user engagement metrics such as completion, likes, and comments. However, these metrics are noisy signals of true latent user utility. A user may become more selective if the perceived recommendation policy delivers higher mean utility or higher variance. To causally test these relations, I build a toy recommender system for short videos using the KuaiRec dataset, which contains large-scale user–video interaction logs together with rich user, item, and context features. I first train a graph neural network (GNN) to predict multiple engagement “heads” (e.g., completion, long watch, rewatch, negative feedback) that are combined into a value model commonly used in recommender systems. I then estimate the head weights by matching the embeddings of simulated recommendations with those of the observed platform recommendations. This toy recommender system leverages the rich feature set and is intended to mimic the deterministic part of the real-world recommendation policy, so that the residual discrepancy between observed and simulated recommendations can be treated as an exogenous shock and used for causal inference. In later work, I will debias these engagement signals to better align them with true user utility, disentangling them from irrelevant factors such as users’ beliefs and search costs.

1 Introduction

Modern recommender systems for short-video platforms, such as TikTok or Kuaishou, are typically optimized using observable engagement metrics such as whether a user completes a video, clicks the like, or leaves a comment. These metrics are easy to log and scale to billions of interactions, but they are only noisy proxies for the true latent utility that users derive from content. In particular, engagement can be influenced by many factors that are not directly about enjoyment, such as users’ prior beliefs about the recommendation quality or their search patterns. Understanding how these engagement signals relate to underlying utility, and how user selectivity changes when the recommendation policy varies in terms of mean utility or risk, is important both for improving user experience and for designing more principled learning objectives for recommender systems.

In this project, I build a toy recommender system for short videos using the KuaiRec dataset, which provides large-scale user–video interaction logs together with rich user, item, and session-level features. My goal is not to fully reproduce a production system, but to approximate the deterministic part of the platform’s ranking policy closely enough that the remaining discrepancy can be interpreted as an exogenous shock. This approximation will later allow me to study how users adjust their selectivity in response to changes in the distribution of the recommended video utility.

The core supervised learning task in this project is *multi-label engagement prediction* at the user–video edge level. The input to my algorithm is a set of interaction records, where each record corresponds to a candidate video in a user session and is represented by: a user identifier, a video identifier, play duration and a vector of features describing the user, the video, and the session context (for example, user history statistics, item category indicators, and basic temporal features). I construct a bipartite graph whose nodes are users and videos, whose

edges represent these candidate interactions, and whose edge attributes are the corresponding feature vectors. I then use a graph neural network (GNN) defined on this bipartite graph to output, for each edge, a vector of four predicted probabilities corresponding to different engagement “heads”: completion, long watch, rewatch, and negative feedback.

These four heads are combined into a scalar value model that scores each candidate video within a session. Formally, if $\hat{y}_{n,h}$ denotes the predicted probability for head h on edge n and w_h is the weight assigned to that head, the value model takes the form $\hat{v}_n = \sum_{h=1}^4 w_h \hat{y}_{n,h}$. Given these scores, the toy recommender ranks candidate videos within each session and simulates the top-ranked recommendations. I estimate the head weights w_h by matching session-level embeddings of the simulated recommendations to those of the observed platform recommendations, so that the simulated policy closely mimics the platform’s deterministic component. The residual difference between observed and simulated recommendations can then be treated as an exogenous shock in downstream analyses.

Within the scope of CS230, the focus of this project is on building and evaluating the predictive model: I compare a logistic regression baseline, an edge-feature-only multilayer perceptron, and the GNN described above in terms of their ability to predict the four engagement heads on held-out data. In later work beyond this class, I plan to use the trained value model and the estimated head weights as building blocks for debiasing engagement signals, with the aim of aligning them more closely with true user utility and disentangling them from confounding factors such as users’ beliefs and their search costs.

2 Related work

Modern industrial recommender systems are dominated by large-scale deep learning models trained directly on engagement proxies such as clicks and watch time. YouTube’s two-stage architecture with deep neural networks for candidate generation and ranking is a canonical example of this approach and shows how deep models can dramatically improve engagement when trained on massive logs [1]. Wide & Deep learning similarly combines a linear “memorization” component with a deep neural “generalization” component to achieve strong online gains on Google Play [2]. These methods are clever in how they balance memorization and generalization at scale, but they largely treat engagement as the ground-truth objective, without explicitly modeling how it relates to latent user utility. A parallel line of work uses graph neural networks to exploit the bipartite user–item structure in recommendation data. PinSage demonstrates that localized graph convolutions on user–item graphs, combined with node features, can scale to web-scale recommendation and outperform traditional matrix factorization [3], and surveys of GNN-based recommender systems document consistent gains over MLP baselines when models are carefully tuned [4]. My project follows this GNN-based paradigm but differs in focusing on multi-head engagement prediction (completion, long watch, rewatch, and negative feedback) rather than a single relevance label, and on combining these heads into a value model with learned weights. Finally, there is extensive work on biases in recommendation logs and methods to debias training and evaluation. Surveys and counterfactual learning-to-rank methods show how propensity weighting and causal ideas can correct for exposure and position bias in implicit feedback [5, 6]. These approaches mainly debias the exposure process, while still treating the corrected engagement signal as a proxy for reward. In contrast, my work starts from a fully observed short-video dataset (KuaiRec) [7] and targets the mapping from engagement to latent value itself, using a toy GNN-based system to approximate the platform’s deterministic ranking component and to set up later debiasing of engagement signals toward user utility.

3 Dataset and Features

I use the KuaiRec dataset, a fully observed user–item interaction dataset collected from the Kuaishou short-video platform. Following the original paper and documentation, I work with the large interaction log (`big_matrix.csv`) together with user- and item-side feature files (`user_features.csv`, `item_daily_features.csv`, and caption-based category metadata). Each row in the final dataset corresponds to a user–video interaction with a timestamp and raw watch-time information, from which I construct four binary engagement heads: `completion`, `long_watch`, `rewatch`, and `negative_feedback`. These four labels form the multi-head output of my model.

The raw data are organized into three high-traffic “bursts” in July–September 2020. Within each burst, I perform a time-respecting train/validation/test split by calendar day, as summarized in Table 2. Aggregating across the three bursts yields roughly 0.8 million training interactions and about 0.2 million interactions in each of the validation and test sets.

Features are built by merging user-, item-, context-, and history-level information into a single interaction-level table. User features include activity degree, live-streaming and author indicators, follow/fan/friend counts and their log-transformed versions, and a set of encrypted one-hot attributes. Item features include video duration, aspect ratio, upload type, visibility status, author and music identifiers, and three levels of content category. Context features include time-of-day (sine and cosine of local hour) and a weekend indicator. Finally, I compute a collection of history features, such as exponential moving averages of past engagement outcomes, category-level EMAs and entropy, author recency and previous completion, and basic session statistics. All numeric features are standardized using the mean and standard deviation computed on the training interactions only. The full list of features used as GNN edge attributes is reported in Table 2-5 in Appendix.

4 Methods

4.1 Problem formulation

Let n index user–video interactions. For each interaction I construct a feature vector $x_n \in \mathbb{R}^d$ and a 4-dimensional binary label

$$y_n = (y_{n,\text{complete}}, y_{n,\text{long}}, y_{n,\text{rewatch}}, y_{n,\text{neg}}) \in \{0, 1\}^4.$$

The four heads are defined from watch behavior as follows:

- $y_{n,\text{complete}} = 1$ if the user watched (almost) the full video.
- $y_{n,\text{long}} = 1$ if the interaction is a long watch (e.g., watch ratio ≥ 1.5 or play duration > 12 seconds).
- $y_{n,\text{rewatch}} = 1$ if the user effectively rewatches the content (e.g., watch ratio ≥ 2).
- $y_{n,\text{neg}} = 1$ if the interaction is negative (e.g., very short watch such as play duration < 2 seconds).

The goal is to learn a function f_θ that maps x_n (plus graph structure) to predicted engagement probabilities

$$\hat{p}_n = f_\theta(x_n) \in (0, 1)^4,$$

which are later combined into a scalar value model for ranking.

I represent the data as a bipartite graph $G = (V_u, V_i, E)$ with user nodes V_u , item (video) nodes V_i , and edges $E \subseteq V_u \times V_i$ corresponding to candidate interactions. Each edge $e_n = (u_n, i_n)$ carries the feature vector x_n and label y_n .

4.2 Baselines

As non-graph baselines I train:

- A logistic regression model that takes x_n as input and predicts each head independently (four one-vs-rest classifiers).
- An edge-feature multilayer perceptron (EdgeMLP) that maps x_n to 4 logits via a small feedforward network with ReLU activations, followed by a sigmoid to obtain $\hat{p}_n \in (0, 1)^4$.

Both models operate only on edge features and ignore user–item graph structure. The same EdgeMLP architecture is used as the readout layer in the GNN, so performance differences can be attributed to message passing rather than extra capacity in the prediction head.

4.3 GNN architecture

The main model is a graph neural network that combines node embeddings, graph convolution, and edge features (Figure 1).

Node embeddings and graph convolution. Each user $u \in V_u$ and item $i \in V_i$ is associated with a learnable embedding vector $h_v^{(0)} \in \mathbb{R}^H$. I apply three layers of graph convolution of the GCN type:

$$h_v^{(\ell+1)} = \sigma \left(\sum_{w \in \mathcal{N}(v)} \alpha_{vw}^{(\ell)} W^{(\ell)} h_w^{(\ell)} \right), \quad \ell = 0, 1, 2,$$

where $\mathcal{N}(v)$ is the set of neighbors of v , $W^{(\ell)}$ are learnable weight matrices, $\alpha_{vw}^{(\ell)}$ are normalization coefficients based on node degrees, and σ is the ReLU nonlinearity. After three layers I obtain final node representations $h_v^{(3)}$ that encode information about the local user–item neighborhood.

Edge-level prediction. For each edge $e_n = (u_n, i_n)$ I look up the user and item embeddings $h_{u_n}^{(3)}$ and $h_{i_n}^{(3)}$ and concatenate them with the edge feature vector:

$$g_n = [h_{u_n}^{(3)} \parallel h_{i_n}^{(3)} \parallel x_n].$$

This combined representation is fed into an MLP:

$$h_n^{(1)} = \sigma(W_1 g_n + b_1), \quad h_n^{(2)} = \sigma(W_2 h_n^{(1)} + b_2), \\ z_n = W_3 h_n^{(2)} + b_3 \in \mathbb{R}^4, \quad \hat{p}_n = \sigma(z_n),$$

where z_n are the logits for the four heads and \hat{p}_n are the predicted engagement probabilities.

4.4 Training objective and optimization

All models are trained as multi-label classifiers with a shared loss over the four heads. Given a batch \mathcal{B} of edges, the loss is the average binary cross-entropy:

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{n \in \mathcal{B}} \sum_{h=1}^4 [-y_{n,h} \log \hat{p}_{n,h} - (1 - y_{n,h}) \log(1 - \hat{p}_{n,h})].$$

For the GNN, I sample random mini-batches of edges from the training set, perform forward propagation through the graph and edge MLP, compute $\mathcal{L}(\theta)$, and update parameters using the Adam optimizer with early stopping based on validation AUC. Evaluation is performed per head and averaged across heads.

4.5 Value model and embedding matching for head weights

The GNN outputs a four-dimensional engagement vector for each edge. To use these predictions for ranking within a session s , I define a linear value model

$$\hat{v}_n(w) = \sum_{h=1}^4 w_h \hat{p}_{n,h},$$

where $w = (w_1, \dots, w_4)$ are nonnegative head weights. For each session s , I simulate a toy recommender that ranks all candidate videos by $\hat{v}_n(w)$ and selects the top K items. Let $\mathcal{S}_s^{\text{obs}}$ denote the set of videos actually shown by the platform in session s and $\mathcal{S}_s^{\text{sim}}(w)$ the set selected by the toy recommender. To compare these two recommendation sets at the representation level, I construct simple *session embeddings* by averaging edge-level representations. For a fixed choice of edge representation ϕ_n (for example, the penultimate MLP layer or the vector of predicted heads), I define

$$\mu_s^{\text{obs}} = \frac{1}{|\mathcal{S}_s^{\text{obs}}|} \sum_{n \in \mathcal{S}_s^{\text{obs}}} \phi_n, \quad \mu_s^{\text{sim}}(w) = \frac{1}{|\mathcal{S}_s^{\text{sim}}(w)|} \sum_{n \in \mathcal{S}_s^{\text{sim}}(w)} \phi_n.$$

The head weights w are then estimated by minimizing an embedding-matching loss over a set of held-out sessions \mathcal{S} :

$$L_{\text{emb}}(w) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \|\mu_s^{\text{obs}} - \mu_s^{\text{sim}}(w)\|_2^2.$$

Intuitively, this procedure chooses head weights so that the distribution of simulated recommendations (in the embedding space) is as close as possible to the distribution of observed recommendations produced by the real platform. The resulting value model is then used as a deterministic approximation of the platform’s ranking rule in subsequent analyses.

5 Experiments, Results, and Discussion

Metrics. For each interaction–video pair and head $h \in \{\text{complete, long, rewatch, neg}\}$ the model outputs a probability $\hat{p}_{n,h} \in (0, 1)$ and is evaluated as a binary classifier. The primary metric is AUC, computed from the ROC curve of $\text{TPR}(t) = \text{TP}(t)/(\text{TP}(t) + \text{FN}(t))$ versus $\text{FPR}(t) = \text{FP}(t)/(\text{FP}(t) + \text{TN}(t))$ as the threshold t varies. I report per-head AUCs on train/validation/test and their mean across heads (Table 1). For the final GNN model I also plot a micro-averaged ROC curve (Figure 4) on the test set by pooling all (n, h) pairs into one list of predictions, which summarizes overall ranking quality across all four heads. Accuracy is less informative because of class imbalance (e.g. only about 7% of interactions have a rewatch), but I confirmed that precision and recall at a fixed threshold are reasonable for all heads.

Hyperparameter selection. I first tune optimization hyperparameters on a “tiny” graph that contains 10% of edges but all node features. Figure 2 shows validation mean AUC for a 64-dimensional GNN trained for 500 epochs with four learning rates $\eta \in \{10^{-4}, 3 \cdot 10^{-4}, 10^{-3}, 3 \cdot 10^{-3}\}$ and batch size 4096. Very small steps (10^{-4}) learn slowly and plateau near 0.75 AUC, while $\eta = 10^{-3}$ and $3 \cdot 10^{-3}$ reach ≈ 0.84 AUC within 150–200 epochs. The largest rate is slightly noisier and does not improve the final AUC, so I fix $\eta = 10^{-3}$. With this choice, a batch-size sweep $B \in \{1024, 2048, 4096, 8192\}$ (Figure 3) shows that $B = 1024$ converges noticeably slower and to a lower AUC, whereas $B \geq 2048$ give almost identical curves. I therefore use $B = 4096$ in all subsequent experiments as a good trade-off between stability and memory use.

Model comparison. Using these hyperparameters, I train four models on the full three-burst dataset with early stopping (patience = 20 epochs based on validation AUC for each head): logistic regression on edge features, an edge-level MLP, and two GNNs with 64 and 128 hidden units. Table 1 shows that logistic regression already performs well (mean test AUC ≈ 0.82), the MLP provides a clear gain (mean test AUC ≈ 0.85), and the 64-dimensional GNN further improves to ≈ 0.848 , with the largest gains on the long-watch and rewatch heads. The 128-dimensional GNN slightly increases train AUC but does not improve validation or test AUC, suggesting diminishing returns from additional capacity. Overall, train and test AUCs are close (e.g. 0.856 vs. 0.848 for the 64-dim GNN), indicating only mild overfitting; early stopping on validation AUC is sufficient to keep generalization gaps small.

Table 1: Per-head AUC for all models on train / validation / test splits.

Model	Train AUC				Valid AUC				Test AUC			
	comp	long	rewatch	neg	comp	long	rewatch	neg	comp	long	rewatch	neg
Logistic regression	0.8268	0.7703	0.8304	0.8588	0.8239	0.7697	0.8307	0.8573	0.8297	0.7718	0.8315	0.8617
Edge MLP (hidden = 128)	0.8498	0.8232	0.8564	0.8654	0.8442	0.8183	0.8521	0.8614	0.8485	0.8198	0.8537	0.8658
GNN (hidden = 64)	0.8548	0.8317	0.8702	0.8677	0.8454	0.8189	0.8545	0.8625	0.8492	0.8213	0.8566	0.8657
GNN (hidden = 128)	0.8514	0.8302	0.8685	0.8692	0.8411	0.8160	0.8511	0.8614	0.8446	0.8194	0.8541	0.8649

Qualitative observations. Across all models, completion and negative feedback are easier to predict than long watching and rewatching, which are rarer and more context dependent. The fact that GNNs help most on these harder heads supports the hypothesis that propagating information over the user–item graph and incorporating author history adds useful structure beyond static user and video features. At the same time, the remaining gap between train and test performance suggests that a substantial amount of variation in engagement is left unexplained, which is exactly the source of exogenous residual variation that later causal analyses will exploit.

Head weight estimation. Finally, I estimate the value-model weights over heads by matching session-level 128-dim embeddings of observed and simulated recommendations using the best GNN model. I minimize an embedding-matching loss with BFGS, which converges in 73 iterations to a final loss of 203.45. The estimated weights are approximately (0.22, 0.10, -0.02 , -0.66) for (complete, long, rewatch, neg), indicating that completion and long watch are valued positively, rewatch receives almost no weight, and explicit negative feedback is heavily penalized.

6 Conclusion and Future Work

I built a short-video recommender on KuaiRec and compared logistic regression, an edge MLP, and GNNs. The 64-dimensional GNN achieved the best mean test AUC, slightly improving over the MLP and logistic baseline, suggesting that modeling user–item graph structure helps most on harder heads (long and rewatch). A simple embedding-matching step then produced value weights that reward completion/long watch and penalize negative feedback. Future work will estimate a structural behavioral model to recover user tastes and a selectivity index. I will then run a causal regression with estimated selectivity on the left-hand side and the exogenized residual between the toy recommender and the observed ranking on the right-hand side. Significant effects would motivate debiasing the engagement signal so that the value model depends only on latent utility, net of irrelevant factors such as user beliefs and search patterns.

7 Contributions

This is solo work and forms a crucial part of one of my research projects. I thank TA Yash Suvidh Kankariya for helpful feedback and suggestions. The code is publicly available at <https://github.com/HaozhanGao0904/CS230-final-project>.

References

- [1] Covington, P. Adams, J. & Sargin, E. (2016) *Deep Neural Networks for YouTube Recommendations*. Proceedings of the 10th ACM Conference on Recommender Systems (RecSys).
- [2] Cheng, H. et al. (2016) *Wide & Deep Learning for Recommender Systems*. Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS).
- [3] Ying, R. et al. (2018) *Graph Convolutional Neural Networks for Web-Scale Recommender Systems*. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD).
- [4] Wu, L. et al. (2022) *Graph Neural Networks in Recommender Systems: A Survey*. ACM Transactions on Recommender Systems.
- [5] Chen, L. et al. (2020) *Bias and Debias in Recommender System: A Survey and Future Directions*. ACM Transactions on Information Systems.
- [6] Joachims, T. et al. (2017) *Unbiased Learning-to-Rank with Biased Feedback*. Proceedings of the 10th ACM International Conference on Web Search and Data Mining (WSDM).
- [7] Gao, H. et al. (2022) *KuaiRec: A Fully-Observed Dataset for Recommender Systems*. arXiv preprint arXiv:2202.10842.

Appendix: Tables and Figures

Table 2: Calendar bursts and train/validation/test splits.

Burst	Calendar range	# days	Train days (earliest)	Validation days	Test days (latest)	Split rule
1	2020-07-05 – 2020-07-12	8	first 4 days	next 2 days	last 2 days	4 / 2 / 2 over 8 days
2	2020-08-01 – 2020-08-10	10	first 6 days	next 2 days	last 2 days	6 / 2 / 2 over 10 days
3	2020-08-27 – 2020-09-05	10	first 6 days	next 2 days	last 2 days	6 / 2 / 2 over 10 days

Table 3: User-level features (prefix u_).

Feature	Description
u_user_active_degree	Categorical user activity level (for example, low / medium / high).
u_is_lowactive_period	Indicator if the user is in a low-activity period.
u_is_live_streamer	Indicator if the user has ever done live streaming.
u_is_video_author	Indicator if the user is also a content creator (has uploaded videos).
u_follow_user_num	Number of users this user follows.
u_follow_user_num_range	Binned range of u_follow_user_num.
u_fans_user_num	Number of followers (fans) this user has.
u_fans_user_num_range	Binned range of u_fans_user_num.
u_friend_user_num	Number of friends (mutual relationships).
u_friend_user_num_range	Binned range of u_friend_user_num.
u_register_days	Days since user registration at the time of interaction.
u_register_days_range	Binned range of u_register_days.
u_onehot_feat0-17	Encrypted user categorical features.
u_follow_user_num_log1p	Log1p-transformed number of users the user follows.
u_fans_user_num_log1p	Log1p-transformed number of followers (fans).
u_friend_user_num_log1p	Log1p-transformed number of friends.
u_register_days_log1p	Log1p-transformed days since registration.

Table 4: Item-level features (prefix i_).

Feature	Description
i_aspect_ratio	Video aspect ratio (height / width).
i_author_id	Encoded ID of the video’s author.
i_video_type	Encoded video type (for example, normal / live / other).
i_upload_type	Encoded upload type (for example, original / re-upload).
i_visible_status	Encoded visibility status (for example, public / private / limited).
i_music_id	Encoded ID of background music used in the video.
i_video_tag_id	Encoded tag ID associated with the video.
i_video_tag_name	Encoded tag-name category for the video.
i_video_duration	Video duration (seconds).
i_age_since_upload_days	Days since the video was uploaded.
i_cat_level1_id	Encoded top-level content category ID.
i_cat_level2_id	Encoded mid-level content category ID.
i_cat_level3_id	Encoded fine-grained content category ID.

Table 5: Context features (prefix `ctx_` plus `burst_id` and `session`).

Feature	Description
<code>burst_id</code>	Burst window identifier (1st, 2nd, or 3rd data burst).
<code>session</code>	Session ID for the user’s viewing session.
<code>ctx_hour_sin</code>	Sine transform of local watch hour (time-of-day feature).
<code>ctx_hour_cos</code>	Cosine transform of local watch hour (time-of-day feature).
<code>ctx_is_weekend</code>	Indicator if the interaction happens on a weekend.

Table 6: History features (prefix `hist_`).

Feature	Description
<code>hist_ema_y_complete</code>	Per-user EMA of past completion events before this interaction.
<code>hist_ema_y_long</code>	Per-user EMA of past long-watch events before this interaction.
<code>hist_ema_y_rewatch</code>	Per-user EMA of past rewatch events before this interaction.
<code>hist_ema_y_neg</code>	Per-user EMA of past negative-feedback events before this interaction.
<code>hist_ema_watchratio</code>	Per-user EMA of past watch ratios before this interaction.
<code>hist_cat_ema_complete</code>	Per-user and category EMA of past completion events.
<code>hist_cat_entropy_l2</code>	Entropy-based measure of user’s category diversity (L2-regularized).
<code>hist_author_recency_days</code>	Days since the user last watched this author before this interaction.
<code>hist_last_complete_author</code>	Indicator if the last video from this author was completed by the user.
<code>hist_has_author_history</code>	Indicator if the user has any prior interaction history with this author.
<code>hist_prev_sess_len</code>	Length of the previous session for this user.
<code>hist_intersess_gap_h</code>	Time gap (hours) between the previous session end and current interaction.

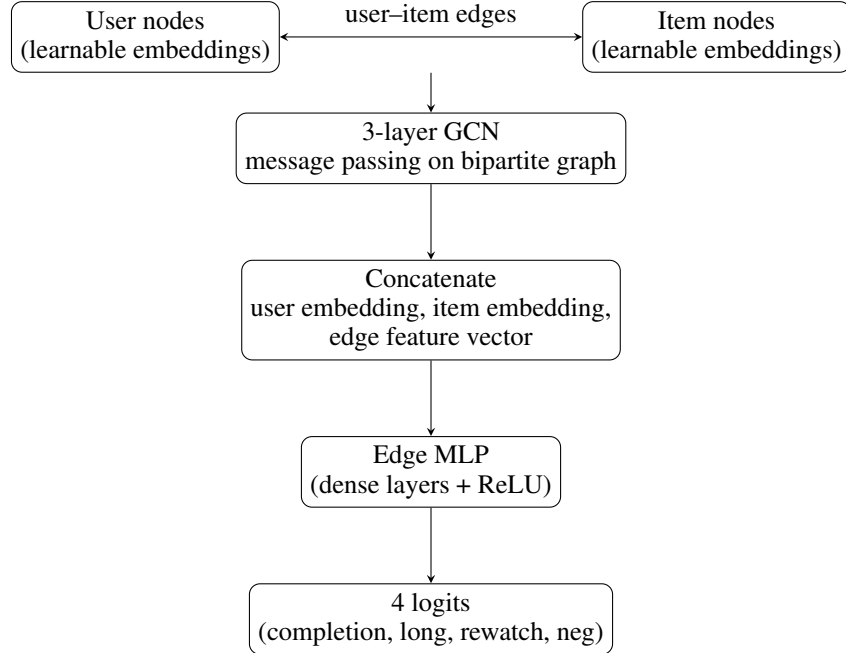


Figure 1: GNN architecture: user and item embeddings are updated by graph convolution, then combined with edge features and passed through an MLP to predict four engagement heads for each edge.

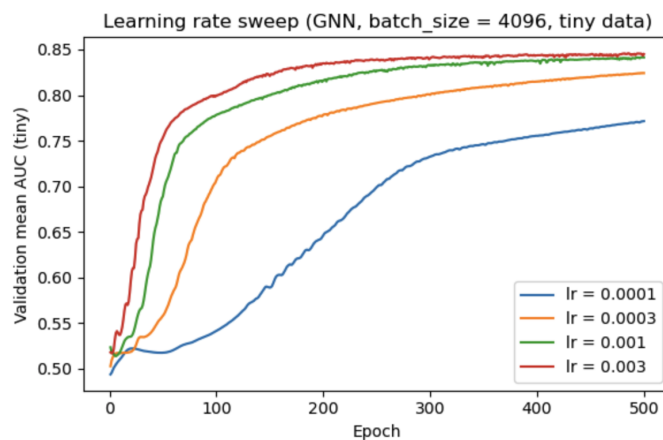


Figure 2: Learning rate sweep for a 64-dimensional GNN (batch size = 4096)

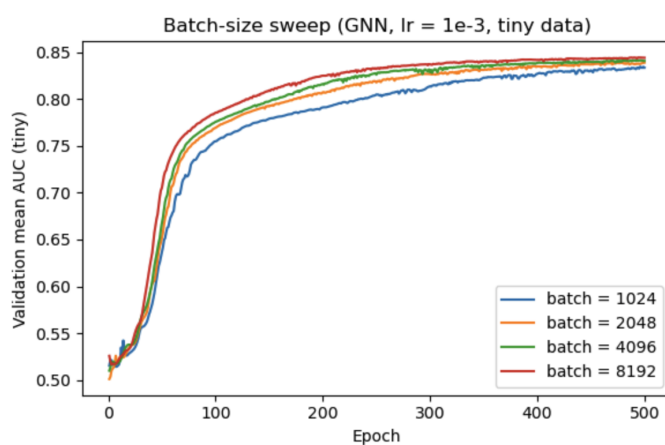


Figure 3: Batch-size sweep for a 64-dimensional GNN (learning rate = 10^{-3})

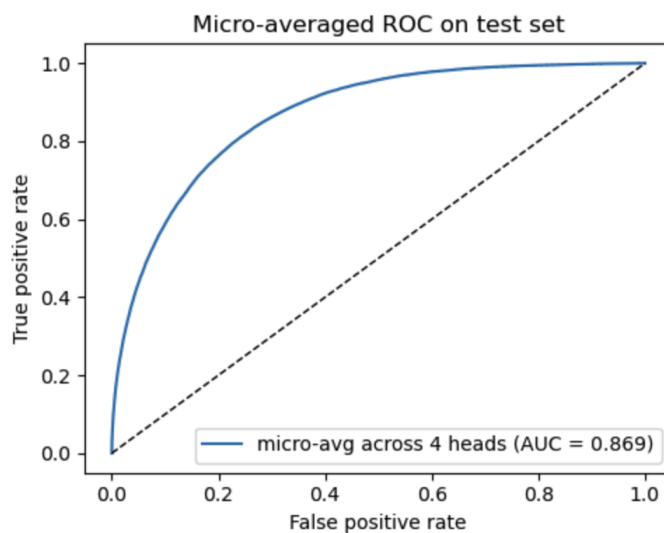


Figure 4: Micro-averaged ROC on test set for a 128-dimension GNN