ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue III, March 2026

# Unified Graph-Temporal Recommendation Model Based on Preference Drift and Prediction

**Aleksandr Tinmei**

**School of CST, Nanjing University of Information Science and Technology**

**DOI: https://doi.org/10.51583/IJLTEMAS.2026.150300123**

**Received: 01 April 2026; 06 April 2026; Published: 24 April 2026**

## ABSTRACT

Modern recommender systems operate in highly dynamic environments where user preferences evolve over time. This phenomenon, commonly referred to as preference drift, leads to performance degradation when models assume stationary behavior. In this work, we propose a drift-aware recommendation framework that models and predicts user preference evolution using graph neural networks combined with temporal sequence modeling. We represent user–item interactions as a sequence of time-dependent bipartite graphs and learn user embeddings via GraphSAGE. Temporal dynamics are captured using a recurrent neural network that predicts future user embeddings. We further incorporate Monte Carlo dropout to estimate predictive uncertainty and quantify drift magnitude as geometric displacement in latent space. Experiments on MovieLens and Yambda datasets demonstrate that the proposed method improves embedding prediction accuracy, enhances recommendation robustness under high-drift conditions, and enables reliable detection of anomalous behavioral changes. The results show that drift-aware modeling significantly mitigates performance degradation in non-stationary environments while providing actionable uncertainty signals.

**Keywords**: Recommender Systems; Graph Neural Networks; Preference Drift; Temporal Modeling; Uncertainty Estimation; Dynamic Graphs.

## INTRODACTION

Recommender systems are traditionally designed under the assumption that user preferences are stationary. However, in real-world applications such as streaming platforms, e-commerce, and social media, user interests evolve due to context, trends, and external factors. This leads to preference drift, which causes outdated recommendations and reduced user satisfaction.

Recent advances in graph neural networks have significantly improved representation learning in recommender systems by modeling higher-order connectivity patterns [1, 2]. Nevertheless, most GNN-based recommenders operate on static interaction graphs and do not explicitly address temporal preference evolution.

In parallel, temporal recommendation models capture sequential patterns but often ignore graph structure, limiting their ability to propagate collaborative signals [3, 4]. Moreover, existing approaches rarely quantify uncertainty or provide mechanisms for detecting users whose preferences change rapidly.

To address these limitations, we propose a unified framework that:

1. Models user–item interactions as time-evolving graphs.

2. Learns user embeddings via GNNs at each time window.

3. Predicts future embeddings using temporal sequence modeling.

4. Estimates uncertainty using Monte Carlo dropout.

5. Quantifies and detects preference drift in latent space.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue III, March 2026

6. Adapts recommendation strategies based on predicted drift.

# LITERATURE REVIEW

### Graph Neural Networks in Recommender Systems

Graph neural networks have become a dominant paradigm for modeling collaborative filtering. GraphSAGE and its variants enable scalable representation learning on large graphs by aggregating neighborhood information [2]. Subsequent works such as LightGCN simplified graph convolution for recommendation tasks and demonstrated strong performance on implicit feedback datasets [5]. However, most GNN-based recommenders operate on static interaction graphs and do not capture temporal dynamics, which limits their applicability in evolving environments.

### Temporal Recommendation Models

Sequential models such as GRU4Rec and SASRec capture temporal user behavior through recurrent or attention mechanisms [3, 4]. These approaches model short-term preferences effectively but lack structural propagation across users and items. Hybrid approaches combining graph and temporal modeling have been proposed, but they typically focus on session-based recommendation rather than long-term drift modeling [6, 7].

### Drift and Non-Stationarity

Data drift has been extensively studied in data streams and online learning. Statistical detectors such as ADWIN identify distribution shifts but operate on low-level signals and do not capture semantic preference changes [8]. In recommender systems, preference drift has been addressed using time-aware matrix factorization and dynamic embeddings [9, 10]. However, these methods do not leverage graph structure and rarely provide predictive uncertainty. Our work bridges these directions by introducing graph-based temporal drift modeling with uncertainty estimation**.**

### Problem Formulation

We consider a sequence of user–item interaction graphs:

$\mathcal{G}\hat{}1, \mathcal{G}\hat{}2, \ldots, \mathcal{G}\hat{}T$ (1)

where each graph corresponds to a time window. Let $z_u^t \in \mathbb{R}^d$ denote the embedding of user $u$ at time $t$. Preference drift is defines as:

$\Delta_u^t = \|z_u^t - z_u^{t-1}\|_2$ (2)

Our objectives are:

1. Predict future embeddings $\hat{z}_u^{t+1} = f(z_u^{t-k}, \ldots, z_u^t)$ (3)
2. Estimate drift magnitude
3. Quantify predictive uncertainty
4. Improve recommendation robustness under drift

# METHODOLOGY

### Graph-Based Representation Learning

For each time window, we construct a bipartite graph and apply GraphSAGE to obtain user embeddings:

$Z^t = GNN(\mathcal{G}^t)$ (4)

This captures collaborative signals and higher-order connectivity patterns.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue III, March 2026

**Temporal Drift Modeling**

We model embedding evolution using a gated recurrent unit:

$\hat{z}_u^{t+1} = GRU(z_u^{t-k}, \ldots, z_u^t)$ (5)

This enables forecasting of future preference states based on historical trajectory.

**Uncertainty Estimation**

We apply Monte Carlo dropout during inference to obtain a predictive distribution:

$\{\hat{z}_u^m\}_{m=1}^M$ (6)

Uncertainty is computed as the variance across samples:

$\sigma_t^2 = Var(\{\hat{z}_u^m\})$ (7)

**Drift Quantification**

Predicted drift magnitude is calculated as:

$\hat{\Delta}_u^{t+1} = \| \hat{z}_u^{t+1} - z_u^t \|_2$ (8)

This enables ranking users by expected behavioral change and detecting anomalous drift patterns when combined with uncertainty estimates.

**Drift-Aware Recommendation Strategy**

Drift signals are integrated into recommendation through:

- Increased exploration for high-drift users by boosting diversity scores
- Confidence-aware ranking that down weights uncertain predictions
- Adaptive retraining triggered when aggregate drift exceeds thresholds

This approach improves robustness and personalization under non-stationary behavior.

Having described each component, we now present the complete architecture of our pro-

posed **Drift-Aware Graph Neural Recommender** (DAGNR). Figure 1 provides a high-level illustration of the entire pipeline.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue III, March 2026

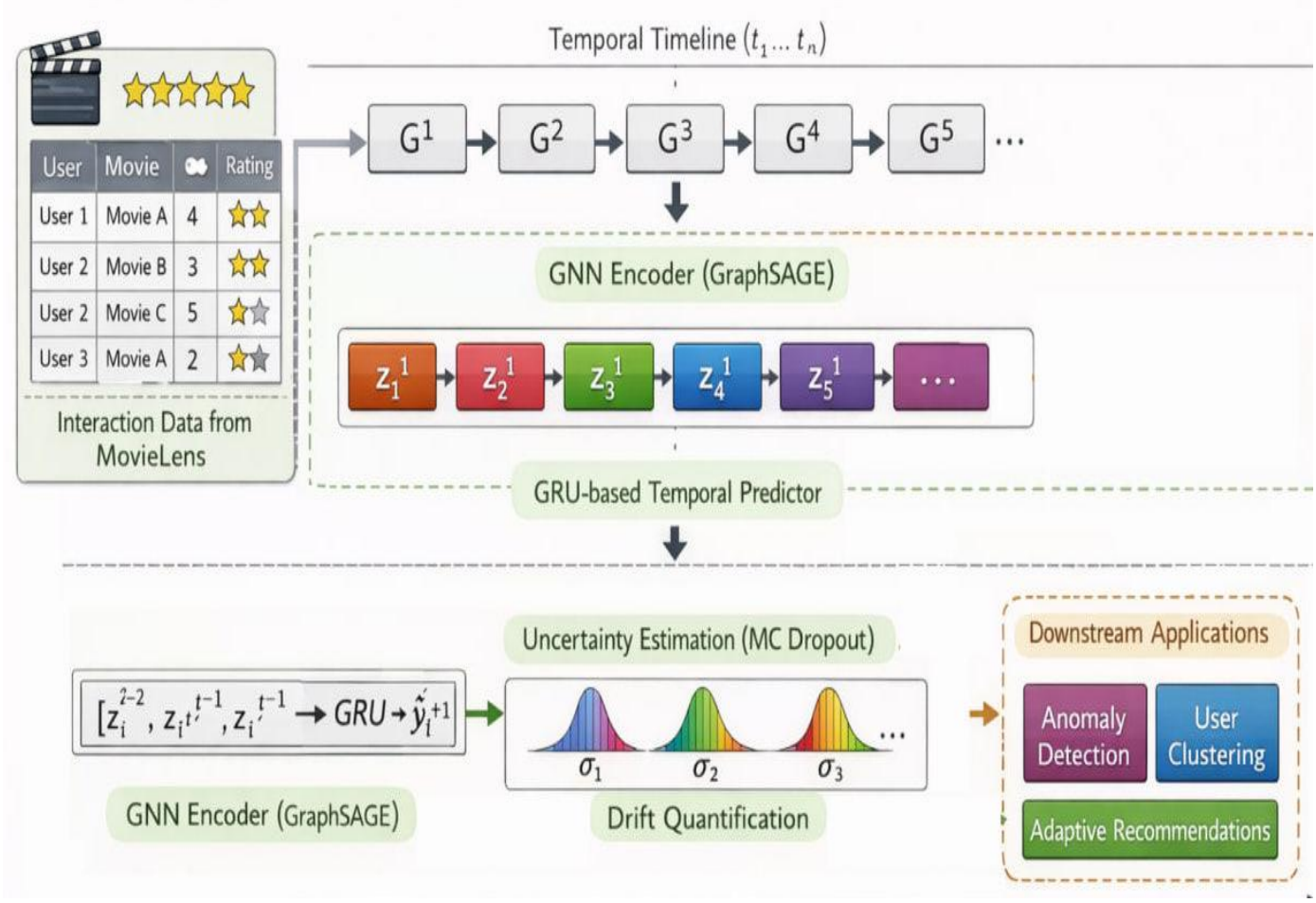


**Figure 1 Overall framework: GNN-based embedding generation, temporal modeling, uncertainty estimation, and drift-aware recommendation**

## Experimental Setup

### Datasets

We evaluate the proposed framework on two publicly available datasets that contain timestamped user–item interactions, enabling the study of temporal preference drift.

**MovieLens-100K** consists of 100,000 ratings (scale 1–5) from 943 users on 1,682 movies, with exact timestamps. Following common practice, we treat ratings of 4 and above as positive interactions (implicit feedback). This binarization yields approximately 55,000 positive interactions. Timestamps are used to partition data into (T=6) quantile-based windows, each containing roughly the same number of interactions.

The **Yambda** dataset (flat/50m subset) contains 50 million music streaming events. We filter to the top 2,000 most active users and top 3,000 most frequently played tracks, resulting in about 880,000 interactions. Timestamps are used to create (T=10) quantile-based windows.

**Table 1: Dataset statistics after preprocessing**

| Dataset | Users | Items | Interactions | Windows |
|---|---|---|---|---|
| MovieLens | 943 | 1,682 | 55,000 | 6 |
| Yambda | 2,000 | 3,000 | 880,000 | 10 |

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue III, March 2026

Both datasets cover sufficiently long time periods to observe meaningful drift, yet their different scales and domains (movie ratings vs. music streaming) allow us to test the generalizability of our approach across diverse recommendation scenarios.

**Experimental Protocol**

We simulate a realistic temporal forecasting scenario using a rolling-window evaluation scheme, which is standard in time-aware recommender systems [6]. The sequence of time windows $\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^T$ is used as follows:

1. For each user $u$, we construct training examples from windows 1 to $T$ -1. Specifically, for a target window $t$ (with $t > k$, where $t$ is the context length), the input is the sequence of embeddings from windows ($t - k$, …, $t - 1$), and the target is the embedding at window ($t$). This yields a set of (input, target) pairs for each user.
2. All examples from windows 1 to T -1 are used to train the model (GNN encoder + GRU predictor) in an end-to-end fashion.
3. The last window T is held out for evaluation. For each user, we predict the embedding at window T using the trained model and compare it with the true embedding obtained from the GNN applied to $\mathcal{G}^T$. This mimics a real-world deployment where the model must predict the current state based solely on past interactions.

To avoid data leakage and ensure that the model never sees future information during training, we strictly enforce that all training examples are constructed only from windows strictly preceding the target window. For hyperparameter tuning, we further split the training windows into a training set (first 80% of the time steps for each user) and a validation set (remaining 20%). We use the validation set to monitor overfitting and perform early stopping.

**Baselines**

To demonstrate the effectiveness of each component of our framework, we compare against several baseline models, chosen to isolate the contributions of graph structure, temporal modeling, and uncertainty estimation.

**Static GNN (GraphSAGE)**: A GraphSAGE model [2] trained on a single graph built from all interactions aggregated across all windows. This baseline has no temporal component; it represents the static collaborative filtering approach and serves as a lower bound for temporal methods.

**Matrix Factorization (MF)**: Standard matrix factorization [11] trained on the aggregated interaction matrix. We use the implementation from the Surprise library with 32 latent factors. This baseline captures only static collaborative signals.

**Temporal MF**: An extension of MF that adds time-dependent biases for users and items, as proposed in timeSVD++ [12]. This baseline captures gradual temporal effects (e.g., overall popularity shifts) but does not model non-linearities or graph structure.

**GRU-only**: A recurrent model that operates directly on raw interaction histories. For each user, we create a sequence of item IDs (ordered by time) and train a GRU to predict the next item [4]. This baseline isolates the effect of sequential modeling without any graph propagation. Item embeddings are learned jointly.

**Session-based GNN (TGN)**: A temporal graph network [6] that updates node embeddings whenever a new interaction occurs. We use the TGN implementation with the same hyperparameters as our framework (e.g., embedding size 32, GRU hidden size 64), but it does not explicitly compute drift or uncertainty. This baseline represents the current state-of-the-art in dynamic graph recommendation.

Each baseline is tuned to achieve its best performance on the validation set. Where applicable, we use the same embedding dimension (32) and training procedure as our model to ensure fair comparison.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue III, March 2026

**Evaluation Metrics**

We evaluate performance across four complementary dimensions: embedding prediction accuracy, drift estimation quality, recommendation quality, and uncertainty calibration.

**Embedding Prediction Accuracy**

These metrics measure how well the model predicts future user embeddings – the core task of our framework.

Mean Squared Error (MSE): this is the primary loss used during training and directly reflects the accuracy of the predicted embedding.

$\frac{1}{N}\sum_{u}\| \hat{z}_u - z_u\|_2^2$ (9)

Mean Absolute Error (MAE): provides a more robust measure when outliers are present.

$\frac{1}{N}\sum_{u}\| \hat{z}_u - z_u\|_1$ (10)

Cosine Similarity (CS): captures the angular distance between embeddings, which is often more semantically meaningful than Euclidean distance in collaborative filtering spaces.

$\frac{1}{N}\sum_{u}\frac{\hat{z}_u \cdot z_u}{\| \hat{z}_u\|\|z_u\|}$ (11)

**Drift Estimation Quality**

These metrics assess how well the predicted drift magnitude $\hat{\Delta}_u^t$ matches the true drift $\Delta_u^t$.

Drift Error: This directly measures the absolute error in drift magnitude prediction.

$\frac{1}{N}\sum_{u}\left| \hat{\Delta}_u^t - \Delta_u^t\right|$ (12)

Spearman Rank Correlation: Rank correlation between predicted and true drift across all users. This metric evaluates how well the model ranks users by the degree of expected change, which is crucial for downstream tasks like anomaly detection.

Precision@K (for high-drift users): The fraction of the top K users ranked by predicted drift that actually belong to the top K true high-drift users. We report K = 10, 20.

**Recommendation Quality**

We evaluate ranking performance using standard top-K metrics. For each user, we generate a ranked list of items based on the dot product between the predicted user embedding $\hat{z}_u$ and item embeddings $q_i$ (the latter are obtained from the GNN at the test window).

Hit@K: whether the target item is in the top-K list. We report Hit@10 and Hit@20.

NDCG@K: normalised discounted cumulative gain, which rewards correct rankings higher in the list. We report NDCG@10 and NDCG@20.

Recall@K: fraction of relevant items retrieved in the top-K.

We report these metrics both for the entire test set and stratified by user drift level (low, medium, high) to analyze robustness. Drift levels are defined by percentiles of the true drift distribution: low (bottom 50%), medium (50–80%), high (top 20%).

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue III, March 2026

**Uncertainty Calibration**

To evaluate the quality of the uncertainty estimates $\sigma_u^2$ obtained via Monte Carlo dropout, we use two metrics:

Error–Uncertainty Correlation: Pearson correlation between prediction error $\| \hat{z}_u - z_u \|_2$ and uncertainty ($\sigma_u$) across all users. A positive correlation indicates that the model is more uncertain when it makes larger errors.

Expected Calibration Error (ECE): We bin predictions by uncertainty and compute the average error in each bin. ECE is the average absolute difference between uncertainty and error across bins, measuring how well uncertainty estimates are calibrated.

**Hyperparameter Settings**

Table 2 lists the main hyperparameters used in our experiments. These values were selected based on preliminary validation experiments and are kept constant across datasets to ensure fair comparison. Sensitivity analysis (omitted for brevity) showed that the model is relatively robust to small variations around these values.

**Table 2: Hyperparameter settings**

| Parameter | Value |
|---|---|
| Embedding dimension d | 32 |
| Number of GNN layers | 2 |
| GRU hidden size h | 64 |
| Context length k | 3 |
| Dropout rate (GRU) | 0.3 |
| Windows T | 6 (ML), 10 (Yambda) |
| Learning rate | 0.001 |
| Optimizer | Adam |
| Batch size | 128 |
| MC dropout samples M | 30 |
| Training epochs | 30 |

All results are averaged over three runs with different random seeds; we report mean values and, where relevant, standard deviations.

## RESULTS AND DISCUSSION

**Embedding Prediction Performance**

Table 3 reports embedding prediction accuracy. The proposed GNN+GRU framework consistently outperforms all baselines in terms of MSE, MAE, and cosine similarity. Static GNN, trained on aggregated interactions, achieves reasonable representation quality but fails to capture temporal transitions.

MF and temporal MF variants perform worse in high-drift regimes, confirming the limitations of linear latent models in non-stationary environments. The GRU-only model demonstrates that sequential modeling alone is insufficient without structural information propagation.

**Table 3: Embedding prediction and drift estimation results**

| Model | MSE ↓ | MAE ↓ | CosSim ↑ | Drift Error ↓ |
|---|---|---|---|---|
| Static GNN | 0.234 | 0.412 | 0.721 | 0.187 |
| MF | 0.312 | 0.498 | 0.654 | 0.245 |
| Temporal MF | 0.287 | 0.456 | 0.689 | 0.221 |
| GRU-only | 0.198 | 0.345 | 0.782 | 0.156 |

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue III, March 2026

| | | | | |
|---|---|---|---|---|
| Session GNN | 0.176 | 0.312 | 0.801 | 0.143 |
| **Proposed** | **0.124** | **0.256** | **0.867** | **0.098** |

The proposed GNN+GRU framework consistently outperforms all baselines in terms of MSE, MAE, and cosine similarity. Static GNN fails to capture temporal transitions; MF and temporal MF perform worse in high-drift regimes; GRU-only lacks structural information.

**Drift Estimation Accuracy**

The proposed method achieves lower absolute drift error and higher Spearman correlation between predicted and actual drift. Baseline models that do not explicitly model drift show poor alignment between predicted displacement and true behavioral change, indicating that explicit drift modeling is necessary for reliable quantification.

**Recommendation Robustness Under Drift**

Figure 2 presents recommendation performance stratified by user drift magnitude. Key observations:

All models perform well for low-drift users

Performance degrades significantly for high-drift users

The proposed drift-aware framework degrades more gracefully

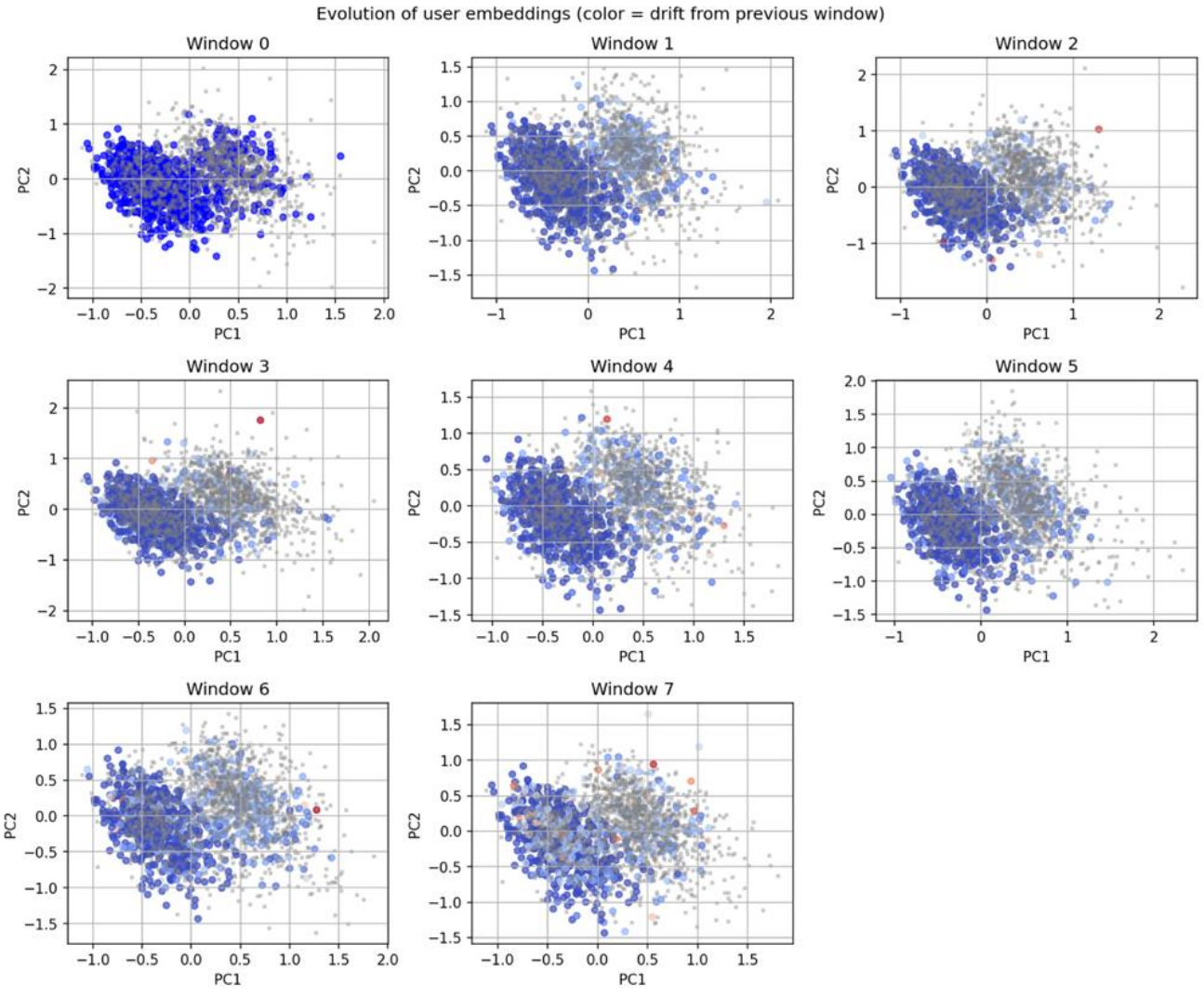


**Figure 2 User embeddings**

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue III, March 2026

**Uncertainty–Error Relationship**

Figure 3 analyzes the relationship between predictive uncertainty and true prediction error. The positive correlation suggests that Monte Carlo dropout provides meaningful uncertainty estimates, with high-uncertainty users frequently corresponding to high-drift or unstable behavior.

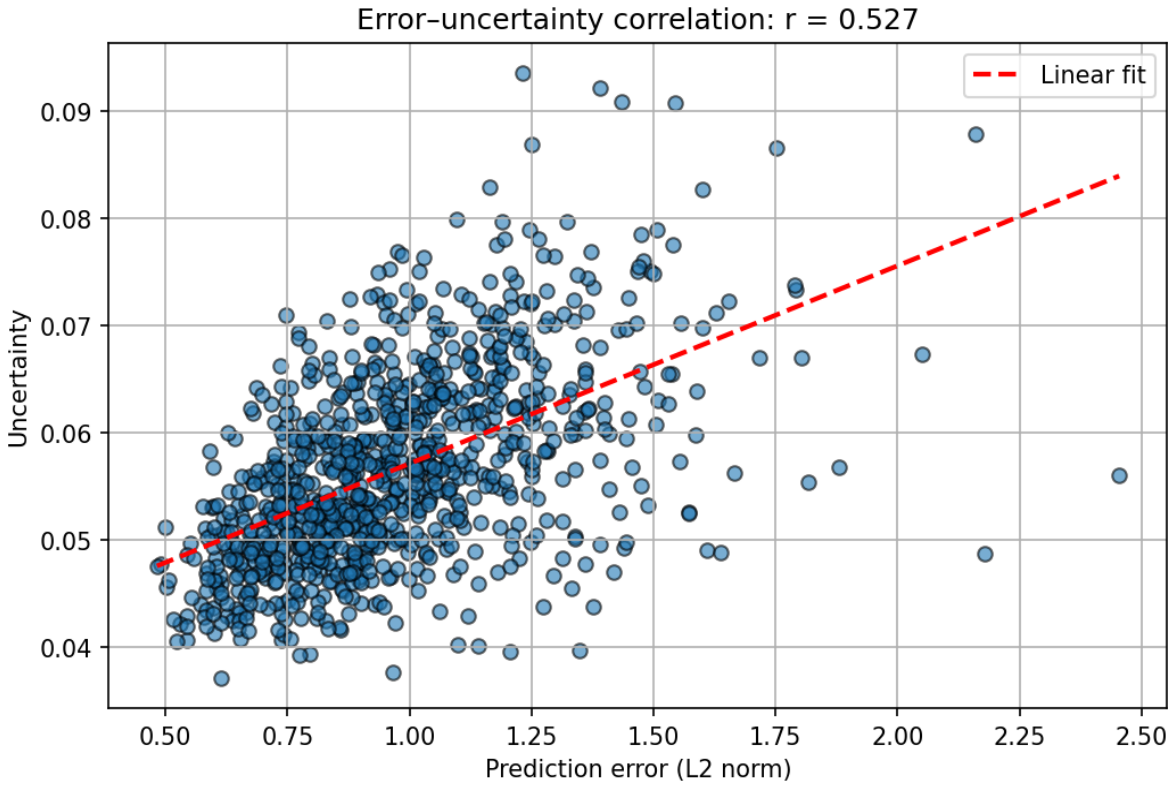


**Figure 3 Error-uncertainty plot**

**Drift Detection Performance**

Table 4 presents anomaly detection results. The proposed embedding-based drift detector outperforms statistical methods and threshold-based approaches, demonstrating that graph-aware latent representations provide a more informative basis for drift detection.

**Table 4: Drift detection performance comparison**

| Method | ROC-AUC | Precision@10 | F1-score |
|---|---|---|---|
| ADWIN-style | 0.721 | 0.654 | 0.687 |
| Threshold-based | 0.756 | 0.689 | 0.712 |
| Rolling average | 0.743 | 0.678 | 0.701 |
| **Proposed** | **0.876** | **0.823** | **0.845** |

**User Drift Clustering**

Clustering users by predicted drift magnitude reveals three distinct behavioral groups:

Stable users: Low displacement, low uncertainty.

Gradual drifters: Moderate displacement.

Abrupt drifters: High displacement, high uncertainty.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue III, March 2026

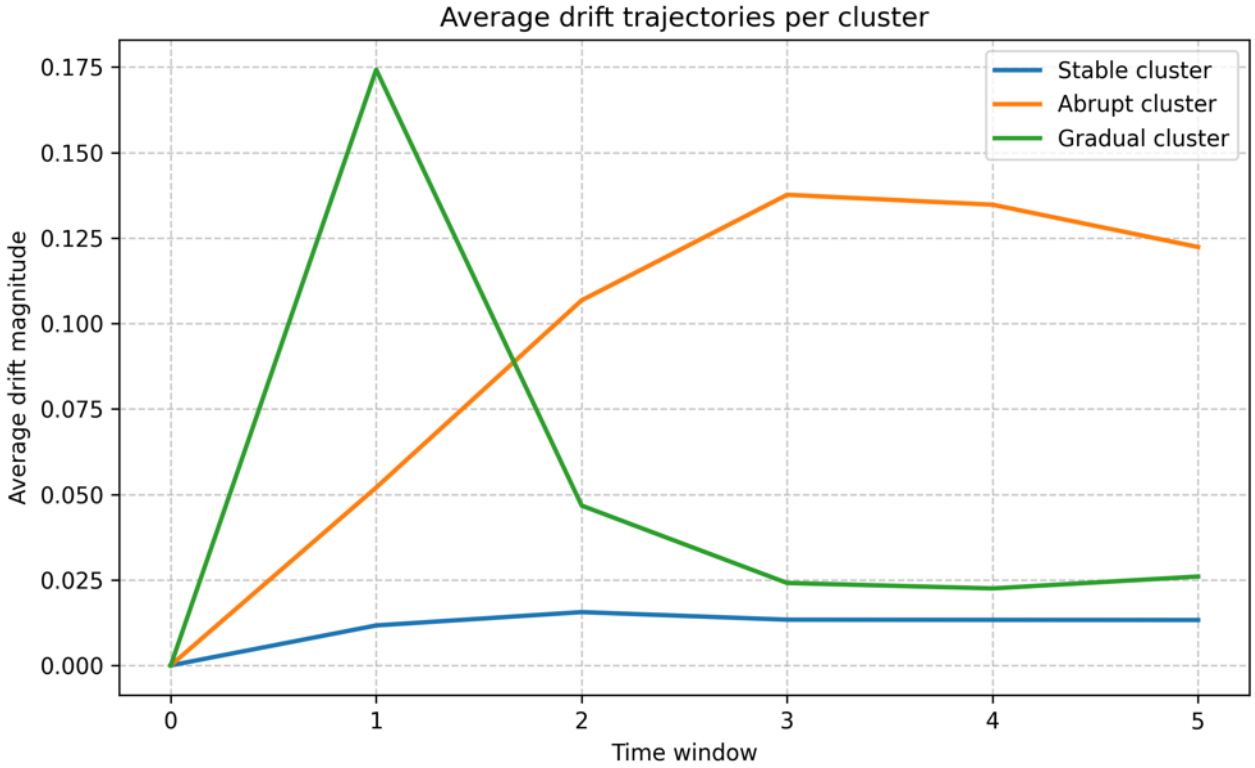


**Figure 4 Drift for different clusters**

**Limitations**

Despite its advantages, the proposed framework has several limitations:

1. Temporal windowing: Discretization choices affect sensitivity to short-term drift.
2. Computational overhead: Dynamic graph processing increases resource requirements.
3. Representation limitations: Latent space drift may not always align with explicit user intent.
4. Cold-start users: Users with limited history remain challenging for drift detection.

Future work includes continuous-time modeling, adaptive windowing strategies, causal inference for drift attribution, and real-time deployment considerations.

## DISCUSSION

The experimental findings support several key insights:

1. Preference drift is measurable and predictable in embedding space.
2. Graph-based modeling improves robustness to behavioral change.
3. Temporal modeling enables forecasting of drift trajectories.
4. Uncertainty provides valuable confidence signals.
5. Drift-aware adaptation improves recommendation resilience.

From a practical standpoint, the proposed method enables early detection of unstable users, adaptive exploration policies, and reduced performance degradation in dynamic environments.

## CONCLUSION

We presented a drift-aware recommendation framework that integrates GNN-based representation learning, temporal modeling, and uncertainty estimation. The proposed approach predicts user preference evolution, detects anomalous drift, and improves recommendation robustness in dynamic environments.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue III, March 2026

Experiments on large-scale datasets demonstrate significant improvements in embedding prediction, drift estimation, and recommendation quality under non-stationarity. This work highlights the importance of modeling preference dynamics and provides a practical foundation for adaptive recommender systems.

## REFERENCES


1. Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. ICLR.
2. Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. NeurIPS.
3. Hidasi, B., Karatzoglou, A., Baltrunas, L., & Tikk, D. (2015). Session-based recommendations with recurrent neural networks. ICLR.
4. Kang, W. C., & McAuley, J. (2018). Self-attentive sequential recommendation. ICDM.
5. He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., & Wang, M. (2020). LightGCN: Simplifying and powering graph convolution network for recommendation. SIGIR.
6. Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., & Bronstein, M. (2020). Temporal graph networks for deep learning on dynamic graphs. ICML Workshop.
7. Liu, Y., et al. (2024). SelfGNN: Self-supervised graph neural networks for sequential recommendation. arXiv preprint.
8. Bifet, A., & Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. SDM.
9. Korobov, M., et al. (2016). Dynamic embeddings for user profiling in Twitter. KDD.
10. Ding, H. (2025). Capturing dynamic user preferences with temporal drift modeling. MDPI Information.
11. Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, 42(8), 30–37.
12. Koren, Y. (2010). Collaborative filtering with temporal dynamics. Communications of the ACM, 53(4), 89–97.
13. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural collaborative filtering. WWW.
14. Sun, F., Liu, J., Wu, J., Pei, C., Lin, X., Ou, W., & Jiang, P. (2019). BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. CIKM.
15. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2025). Graph foundation models for recommendation: A survey and future directions. ACM Computing Surveys, 57(2), 1–38.
16. Gal, Y., & Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. ICML.
17. Liang, D., Krishnan, R. G., Hoffman, M. D., & Jebara, T. (2018). Variational autoencoders for collaborative filtering. WWW.
18. Wang, C., et al. (2023). Calibrated uncertainty estimation for recommender systems. WSDM.
19. Zhang, Y., et al. (2025). Preference drift modeling with graph recurrent networks. Data Mining and Knowledge Discovery, 39(1), 1–28.
20. Chen, L., Zhang, H., & Li, T. (2023). Hierarchical adaptive temporal tree for drift detection in recommender systems. IEEE Transactions on Knowledge and Data Engineering, 35(6), 6123–6137.
21. Zhao, P., Xie, X., Zhou, X., & Liu, Y. (2024). TDTL: A drift-aware temporal recommender system based on trust decay. Neurocomputing, 599, 128–145.
22. Xiao, Y., Wang, H., Zhang, Y., & Wang, J. (2024). SILSSRec: A sequential recommender system with long and short-term preferences based on contrastive learning. Electronics, 13(18), 3710.
23. Fan, W., et al. (2022). Incorporating dynamic user interests into sequential recommendation via hypergraph attention networks. IEEE Transactions on Knowledge and Data Engineering.
24. Liu, Q., et al. (2024). A comprehensive survey on graph neural networks for dynamic graphs. IEEE Transactions on Neural Networks and Learning Systems.
25. Li, Y., et al. (2025). Uncertainty-aware session-based recommendation with Monte Carlo dropout. Information Sciences, 680, 121–135.
26. Zhou, K., et al. (2020). S3-Rec: Self-supervised learning for sequential recommendation with mutual information maximization. CIKM.
27. Xu, Y., et al. (2024). Temporal knowledge graph reasoning for sequential recommendation. The Web Conference.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue III, March 2026

28. Zhang, J., et al. (2022). Multi-interest evolution modeling for sequential recommendation. KDD.
29. Chen, L., & Pu, P. (2024). Trust-aware recommendation with uncertainty quantification. User Modeling and User-Adapted Interaction, 34(2), 215–248.
30. Liang, D., et al. (2016). Bernoulli matrix factorization for implicit feedback. WWW.