Appendix

Summary of Notations

We summarize the major notations used throughout the paper in Table 1.

Table 1: Summary of Notations

Notation	Description
u, i	user and item
z_i, e_i	item i's original encoding and embedding
$\hat{r}^t_{(u,i)}$, $r^t_{(u,i)}$	predicted and ground truth scores for user u on item i at period t
u^t_{ts}, u^t_{te}	time-specific and time-evolving factors of user u at period t
θ^t, θ^t_u	meta and user-specific parameters of time-specific module at period t
ω	parameter of time-evolving module
$\mathcal{S}^t_u$, $\mathcal{Q}^t_u$	support and query sets in task corresponding to user u at period t
D^t_u	items interacted with user u at period t

Experimental Settings

We initialize both meta-learning and recurrent models with random initial values. Model learning rates are set through a grid search, and the Adam optimizer is applied with L2-regularization. In a dynamic meta-learning setting, the recurrent module takes historical interactions up to $t-1$ time period as an input while predicting for the next time period t, but the meta-learning module takes few recent interactions from the current time as a support set (e.g., in our setting, we have three months of period, so it takes K-shot interactions from the first month of the t time period and remaining interactions of two months as query set). We split users into meta-train and meta-test sets. To make the problem more challenging, we consider time-sensitive users with few interactions in the current period as test users. We set the few-shot ($K = 5$) to select the limited interactions for the support set.

For non-meta learning methods, the meta-train and meta-test split does not apply. To make a fair comparison, we first collect user interactions from period 1 to $t-1$ and then take the first few interactions (K) from the current time period t to create the training set. The remaining interactions from time period t are used for the test set. This mimics the few-shot problem for the current time period, and then a few interactions are available for the model to learn the user factors.

Different from explicit ratings that are constrained in a range, some implicit counts could take very large values. As a result, most of the models are not specifically designed for implicit data. To address this issue, we take a logarithm transformation on the Last.fm dataset to make the observed entries more balanced. Since DPF is specifically designed for counts data (only compared on Last.fm), we do not take logarithm transformation during training, but take such transformation on prediction and observation when calculating RMSE to make a fair comparison.

Datasets

The Netflix dataset has around 100 million interactions, 480,000 users, and nearly 18,000 movies rated between 1998 to 2005. We preprocessed the dataset similar to (**?**), which consists of user-item interactions from 01/2002 to 12/2005. Movie attributes are taken from the IMDB website, in which we considered genres, directors, actors, movie descriptions, and overall movie ratings as the key movie attributes. The MovieLens-1M dataset includes 1M explicit feedback (i.e., ratings) made by 6,040 anonymous users on 3,900 distinct movies from 04/2000 to 02/2003. This dataset has very dense ratings in a particular time range. Thus we split datasets in a period of 6-months and got a total of 6 periods in contrast to the other two datasets, which have 16 periods considering a period of 3 months. We use the same preprocessing for this dataset as we did for the Netflix dataset. The Last.fm dataset is created by crawling the user interactions and track details from the Last.fm database. This dataset includes 12,902 unique tracks and 548 users from 01/2012 to 12/2015. Tracks are described with artists, tags, and summary information.

Performance of Individual Modules

The proposed model integrates two modules to capture time-specific and time-evolving latent factors. We study their contribution in detail to show how the user's time-specific interest and time-evolving preferences affect the overall recommendation performance.

- **Time-specific Meta-Learning (TS-ML) Module.** This module is specifically designed to capture users' time-specific interest in each period. Different from (**?**), it only relies on time-specific data so that the proposed model is more generally applicable.
- **Time-evolving RNN (TE-RNN) Module.** This module is designed to capture users' gradual shift of interest by utilizing historical interactions.

Table 2: Comparison of recommendation performance (Average RMSE and NDCG) using each module alone and the proposed integrated model for all three datasets

Dataset	Model	RMSE	NDCG
Netflix	TS-ML	0.9380±0.02	0.3103
	TE-RNN	0.9478±0.03	0.3011
	Proposed	**0.8925±0.03**	**0.3472**
Last.fm	TS-ML	1.2791±0.12	0.3073
	TE-RNN	1.9938±0.34	0.2910
	Proposed	**1.2203±0.16**	**0.3385**
MovieLens-1M	TS-ML	1.0935±0.07	0.3144
	TE-RNN	1.2505±0.13	0.3162
	Proposed	**0.9945±0.08**	**0.3351**

Table 2 reports the recommendation performances of each module and compares them with the proposed integrated model. First, each module performs reasonably well, and the recommendation results are comparable with some of the state-of-the-art baselines. It is also interesting to see that the meta-learning model outperforms the time-evolving module in all cases. This demonstrates the stronger impact of

the time-specific factors when making recommendations to users. It also justifies the proposed meta-learning module's effectiveness that successfully captures these latent factors by learning the global trend from other users during a specific period. Finally, the proposed model that integrates both modules achieves the best recommendation performance, because it can capture both the time-evolving and time-specific factors.

We further provide an illustrative example from the MovieLens-1M dataset for a user with ID:3462 to show each module's contribution to the final prediction. Figure 1 shows that final predicted ratings are the combination of both modules, and they effectively complement each other to provide better final performance.

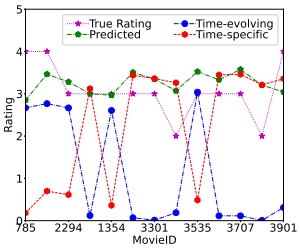


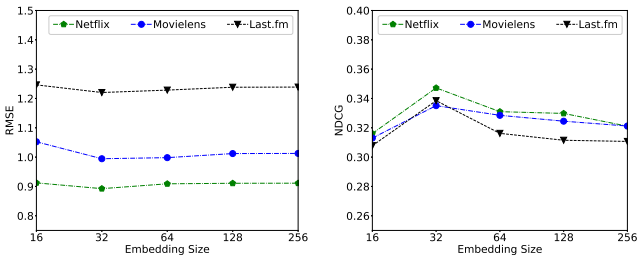
Figure 1: Contribution of time-evolving and time-specific modules to the recommendation of different movies, where they jointly contribute to the predicted ratings.

Varied Recommendation Period Lengths

In this section, we show a more fine-grained temporal analysis by varying the period length. We use three different period lengths: 1 month, 3 months, and 6 months, in the Netflix dataset and report the model performance. We also select one representative from each category of baseline models and report the RMSE and NDCG in Table 3. The user base and the number of interactions per user are limited when we set the length to 1 month. Due to this, matrix factorization methods and deep learning methods suffer largely and have poor performance compared to the proposed model. In the case of 6 months, due to more interactions, deep learning models are performing better than other smaller period lengths. Also, we perform $K = 5$-shots for meta-learning, and due to large interactions present in a 6-month period length, it is quite possible not to get those shots from the most recent interactions. This could hurt the performance of meta-learning, which achieves slightly lower performance than the 3-month period. Overall, in all period lengths, our model outperforms others with a noticeable margin.

No Interactions in the Current Period

We further study the model performance where a user doesn't have any interaction in the current period. In this case, our time-specific module utilizes global user factors instead of user-specific factors due to lack of interaction, and hence no adaptation is made. Similarly, the time-evolving module just forwards its previous evolving user factors to the next time period to provide an integrated recommendation.



(a) RMSE (b) NDCG

Figure 2: Impact of item embedding size

We provide an illustrative example, where we randomly choose five test users with user ids: {870391, 1197396, 757756, 920368, 1918714} and remove their interactions in a given period (i.e., period = 4). We compare the prediction performance with the regular model where these interactions are not removed to show the impact of completely missing interactions in the current period. Table 4 shows that the proposed model provides quite robust results. In Period 4, due to missing interactions, the meta-learning module only utilizes global user factors. Similarly, for Period 5, the time-evolving module doesn't have inputs in Period 4 and hence forwards the user's time-evolving factors from Period 3. The performance with no interactions is slightly worse than having interactions.

Table 4: Results with/without interactions in current period

Period	Interactions		No Interactions	
	RMSE	NDCG	RMSE	NDCG
4	0.8720±0.04	0.3382	0.8953±0.03	0.3226
5	0.8466±0.04	0.3448	0.8569±0.04	0.3415

Hyperparameter Tuning

We perform fine-tuning of model learning rates through a grid search from {1e-5, 1e-4, 1e-3, 1e-2, 1e-1} and select best value as $\alpha = \beta = \gamma = 1e^{-4}$. In our implementation, we learn user embedding from the model but item embedding is generated from the corresponding attributes. We embed each attribute (or feature) of an item in a vector of size 32. Each item in movies datasets has 5 attributes (genre, director, actors, plot, and overall rating), which are concatenated to generate the final item embedding of size 160. We applied different vector sizes [16,32,64,128,256] for each attribute and at 32 we got the optimal performance. Increasing the size didn't improve the results as shown in Figure 2 for three datasets. So, we choose 32 for each attribute and the final embedding size is set as 160.

Periodical Recommendation Results

We report the detailed result for each prediction period for all three datasets to demonstrate how the predictions evolve over time.

Netflix. As can be seen from Table **??**, in most periods, the proposed model performs better than others except for a few periods like 3 and 6, where the proposed model slightly

Table 3: RMSE and NDCG in varying period lengths in months

Model	1		3		6	
	RMSE	**NDCG**	**RMSE**	**NDCG**	**RMSE**	**NDCG**
SVD++	1.0923±0.04	0.2892	0.9797±0.03	0.2915	0.9730±0.05	0.2901
timeSVD++	1.0742±0.04	0.3086	0.9538±0.06	0.3125	0.9641±0.04	0.31142
deepFM	1.1260±0.09	0.2817	0.9811±0.04	0.2930	0.9585±0.06	0.2891
MeLU	1.0037±0.05	0.3138	0.9213±0.05	0.3232	0.9414±0.04	0.3204
Proposed	**0.9864±0.06**	**0.32070**	**0.8925±0.03**	**0.3472**	**0.9226±0.03**	**0.3317**

under-performs the meta-learning model. A possible explanation is that in these periods, time-specific user interest might largely deviate from the time-evolving user factors, and hence their combined recommendation is less accurate.

Last.fm Datasets. The period-wise results for the Last.fm dataset on one run is shown below in Table 6. The proposed model achieves good results in this implicit feedback (i.e., counts) dataset. The high variation of the counts indicates users' music listening habits are fluctuating significantly. The results in Table 6 show that matrix factorization and deep learning baseline models are less effective in capturing those variations. In contrast, the proposed model simultaneously captures those variations in the form of users' specific biases and the gradual shift of preferences effectively.

Movielens Datasets: Periodic results for the Movielens dataset are shown in Table 7. In the first period, we notice that meta-learning models are not performing well, whereas SVD models are performing well. This is because the dataset has very dense interactions in the first period. Meta-learning models only use k-shot for learning, but SVD models benefit from maximum interactions. For the proposed model, time-evolving user factors don't contribute in the first period. Also, time-specific factors are based on meta-learning. Hence, its performance is not better than the baselines, but the proposed model achieves a better performance in the subsequent periods.

Time Complexity

Denote the computational complexity of embedding, time-specific and time-evolving module for one round of back-propagation as $O(m_1)$, $O(m_2)$, $O(m_3)$. In one epoch, the model iterates through T tasks, and in each task, the model performs a local update for the time-specific module on the support set of size S and also updates the time-evolving module on both support S and query set Q. After iterating through the tasks, a meta update is performed utilizing query set Q on the time-specific module. Overall, the total time complexity is $O(m_1 + T(m_2 + m_3)(S + Q))$ for each iteration. We also provide the actual time taken by the proposed model to complete one training iteration utilizing GeForce RTX 2070 GPU. The proposed model considers each user as a task where the time-specific module and time-evolving module take 0.0065s and 0.0023s, respectively, to train on one task. Considering Movielens-1M dataset, we used 80% training users and hence total training time for both modules in each iteration are 31.40s and 11.11s. Similarly, the embedding module takes 3.14s. This gives the total time of 41.65s/iteration.

Link for the Source Codes

https://github.com/ritmininglab/A-Dynamic-Meta-Learning-Model-for-Time-Sensitive-Cold-Start-Recommendations

Table 5: Netflix Periodic RMSE Results

Period	SVD++	timeSVD++	CKF	MeLU	DeepFM	Wide & Deep	GC-MC	Caser	DIEN	ML-ICS	Proposed
1	0.9813	0.9840	0.9552	0.9201	1.2706	1.3178	1.2912	1.2845	1.3102	0.9335	0.9205
2	0.9862	0.9895	0.9683	0.9481	1.1117	1.1321	1.2903	1.2882	1.3033	0.9498	0.9258
3	0.9795	0.9795	0.9693	0.9440	1.0489	1.0921	1.2732	1.2634	1.2724	0.9445	0.9479
4	0.9742	0.9708	0.9524	0.9374	1.0896	1.0777	1.2613	1.2404	1.2543	0.9401	0.8771
5	0.9800	0.9710	0.9432	0.9415	1.0630	1.0115	1.2311	1.2127	1.2167	0.9426	0.9279
6	0.9757	0.9740	0.9564	0.9495	0.9911	0.9904	1.1374	1.1206	1.1515	0.9487	0.9533
7	0.9744	0.9725	0.9612	0.9506	1.0323	0.9895	1.1068	1.1745	1.0984	0.9622	0.8723
8	0.9766	0.9661	0.9526	0.9418	0.9706	1.0122	1.049	1.1132	1.0401	0.9517	0.8789
9	0.9737	0.9603	0.9412	0.9424	0.9902	0.9897	1.0442	1.0724	1.0311	0.9503	0.8786
10	0.9726	0.9597	0.9228	0.9391	0.9961	0.9655	1.0918	1.0843	1.0511	0.9430	0.9209
11	0.9722	0.9595	0.9332	0.9360	0.9497	1.0570	1.03403	1.0563	1.0615	0.9315	0.8997
12	0.9731	0.9511	0.9541	0.9438	0.9761	0.9648	1.0885	1.0245	1.0528	0.9396	0.9427
13	0.9361	0.9556	0.9243	0.9351	0.9710	0.9540	1.0451	1.0373	1.0417	0.9306	0.8746
14	0.9658	0.9431	0.9416	0.9457	0.9720	0.9635	1.031	1.0143	1.0531	0.9487	0.8824
15	0.9673	0.9457	0.9358	0.9378	1.0050	0.9598	1.0162	1.0142	1.0237	0.9441	0.8812
16	0.9670	0.9340	0.9337	0.9428	0.9866	0.9550	1.0254	1.0078	1.0145	0.9468	0.8340

Table 6: Last.fm periodic RMSE results

Period	SVD++	timeSVD++	CKF	DPF	MeLU	DeepFM	Wide & Deep	Caser	DIEN	ML-ICS	Proposed
1	1.6507	1.6920	1.8353	1.6953	1.3153	2.1434	2.1993	1.8132	2.4720	1.3233	1.3173
2	2.3052	2.1647	1.3964	1.3238	1.5567	2.1336	2.1675	1.3854	2.1291	1.5407	1.5320
3	2.1972	1.9160	1.7007	2.0814	1.3408	2.0674	2.1194	1.6772	2.1065	1.3531	1.3465
4	2.1574	1.7385	1.5462	1.4654	1.1346	2.0757	2.0464	1.5232	1.9520	1.1287	1.1047
5	1.5858	1.6520	1.6602	1.5214	1.1687	2.0202	2.0033	1.5843	1.9876	1.1732	1.1578
6	1.7879	1.8460	1.5555	1.3815	1.2602	2.0763	2.1651	1.5278	1.8816	1.2821	1.0809
7	1.5348	1.6498	1.6527	1.3448	1.6359	1.9372	1.9595	1.6227	1.8392	1.6324	1.6226
8	1.7527	1.7350	1.5798	1.3617	1.5987	1.8883	1.8839	1.5482	1.8806	1.6037	1.4270
9	2.2419	1.9977	1.4311	1.5085	1.4493	1.9546	2.0015	1.4326	1.9011	1.4488	1.3338
10	1.9101	1.7901	1.4572	2.3051	1.1753	1.8346	1.8766	1.4104	1.8743	1.1714	1.0368
11	1.5709	1.4704	1.3067	1.4337	1.1528	1.7557	1.8178	1.2974	1.8515	1.1553	1.0864
12	1.5864	1.5760	1.3353	1.2409	1.2226	1.7666	1.7346	1.3136	1.8483	1.2105	1.0933
13	1.5228	1.3637	1.4613	1.0546	1.0546	1.7518	1.7192	1.4463	1.8617	1.0720	1.0123
14	1.7330	1.5966	1.6130	1.2670	1.0809	1.6417	1.6800	1.5812	1.8445	1.0912	1.0603
15	1.9891	1.6644	1.4429	1.3264	1.1094	1.6815	1.7253	1.4293	1.7623	1.0996	1.0824
16	1.4638	1.4106	1.4334	1.2547	1.3206	1.6900	1.9963	1.4017	1.7456	1.3322	1.0688

Table 7: Movielens periodic RMSE results

Period	SVD++	timeSVD++	CKF	MeLU	DeepFM	Wide & Deep	GC-MC	Caser	DIEN	ML-ICS	Proposed
1	1.0615	1.0496	1.1155	1.2234	1.5341	1.3719	2.3345	2.2941	2.3438	1.2412	1.2253
2	0.9954	0.9932	0.9847	1.1613	1.3659	1.2686	1.7912	1.7247	1.7065	1.1803	1.0444
3	1.0445	0.9982	1.0305	0.9341	1.2267	1.2585	1.2576	1.2289	1.2019	0.9423	0.8487
4	1.1499	1.1189	1.0508	0.9776	1.2492	1.2346	1.1332	1.1223	1.1623	0.9711	0.9084
5	1.0918	1.0611	1.1468	1.0308	1.1615	1.1052	1.1346	1.1286	1.1587	1.0514	0.9053
6	1.0773	1.0688	1.0699	1.1377	1.0994	1.0672	1.1112	1.1021	1.1421	1.1127	1.0377