The Limits of Graph Samplers for Training Inductive Recommender Systems

Theis E. Jendal
Aalborg University
tjendal@cs.aau.dk

Matteo Lissandrini
University of Verona
matteo.lissandrini@univr.it

Peter Dolog
Aalborg University
dolog@cs.aau.dk

Katja Hose
TU Wien
katja.hose@tuwien.ac.at

ABSTRACT

Inductive Recommender Systems are capable of recommending for new users and with new items thus avoiding the need to retrain after new data reaches the system. However, these methods are still trained on all the data available, requiring multiple days to train a single model, without counting hyperparameter tuning. In this work we focus on graph-based recommender systems, i.e., systems that model the data as a heterogeneous network. In other applications, graph sampling allows to study a subgraph and generalize the findings to the original graph. Thus, we investigate the applicability of sampling techniques for this task. We test on three real world datasets, with three state-of-the-art inductive methods, and using six different sampling methods. We find that its possible to maintain performance using only 50% of the training data with up to 86% percent decrease in training time; however, using less training data leads to far worse performance. Further, we find that when it comes to data for recommendations, graph sampling should also account for the temporal dimension. Therefore, we find that if higher data reduction is needed, new graph based sampling techniques should be studied and new inductive methods should be designed.

PVLDB Reference Format:
Theis E. Jendal, Matteo Lissandrini, Peter Dolog, and Katja Hose. The Limits of Graph Samplers for Training Inductive Recommender Systems . PVLDB, 18(8): 2496 - 2504, 2025.
doi:10.14778/3742728.3742743

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/GraphRecommendation/gsampling.

1 INTRODUCTION

Recommender Systems (RSs) are used in many applications, ranging from online retail stores to advertisement platforms. These systems utilize historic interactions between users and items to estimate future user behaviors, with the hypothesis that users with similar historical preferences will exhibit similar behavior in the future; often referred to as Collaborate Filtering (CF) [21]. Most approaches capture user preferences and item concepts as dense vector representations, called embeddings, within high-dimensional spaces, such that similar users and items have similar embeddings [21, 46]. To build these representations, deep neural networks learn vector representations assigned to all users and items often through dictionary encodings. These are called transductive techniques [21, 46]. This also means that, when a new user or item is added to the system, in theory they are required to re-train the model to compute the missing embeddings.

Recently, a lot of focus has been placed on inductive RSs due to their ability to predict for unseen users and items [21, 46, 49, 50, 52]. These systems do not learn a unique vector for each user and item but instead learn to generate vectors based on their features and connections. A transductive RS would not be able to recommend "The Dark Knight" in Figure 1 as it is not present in the train graph. In contrast, an inductive RS can recommend items (and to users) absent during training but introduced at inference time [21, 25, 40, 45, 46]. Among inductive methods, only a few can recommend effectively for both new users and items (see Table 1). Yet, the training time of these inductive methods can be very slow, taking up to *2 days to train* on a Collaborative Graph (CG) with $\sim 175k$ users and $\sim 77k$ products. Such long training times are particularly impactful for hyperparameter tuning, where multiple training cycles are often required. Therefore, recent works study how to sample training data to decrease tuning time [12, 32]. They focus on hyperparameter tuning, perform random sampling, and, most importantly, still require the methods to train on the full data afterwards. This is particularly limiting if we consider that the graphs continuously evolve, with millions of items being added each day in some cases [30].

In the past, graph sampling has been proven effective for studying important graph properties on a smaller scale [26]. Thus, in this paper, we are interested in studying whether it is possible to utilize graph sampling approaches to reduce the computational cost and, hence, the training time in the training step of graph-based RSs. Since inductive methods are capable of predicting for new users and items, we, in theory, do not require any retraining of the methods on the full dataset to be able to perform inference on it and thus to recommend new items or to new users. When performing graph sampling, current methods only sample within each batch, reducing batch forward propagation time but maintaining the number of batches in an epoch [31]. Therefore, they still go through all available graph data. Instead, by subsampling the graph *before* training, we effectively obtain a smaller graph structure and thus reduce the training data that needs to be processed. We are

Proceedings of the VLDB Endowment, Vol. 18, No. 8 ISSN 2150-8097.
doi:10.14778/3742728.3742743

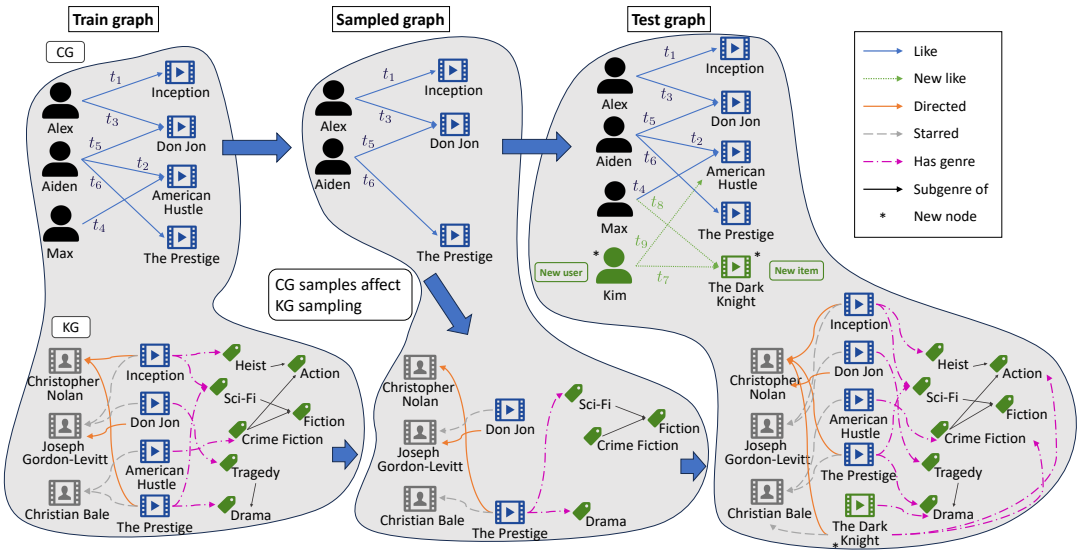

Figure 1: Sampling example with illustration of sampling graph and the correlation between CG samples and KG samples.

interested in studying how to find a suitable subset of training nodes, that allows us to obtain an induced subgraph that is representative enough for the models to learn an inductive bias suitable for future recommendation. The objective being to reduce training time with as little impact on final predictive quality as possible. While the graph-sampling literature has proposed many different techniques [26, 31], these techniques have not been studied with graph neural networks, the current de-facto standard architecture in RSs. Further, only one sampling method (node-based random sampling) and RS (PinSAGE [49]) has been tested directly on just a sub-sampled graph. That is, the more established graph sampling techniques have yet to be tested in this domain. We therefore study three state-of-the-art inductive RSs on three real-world datasets using six graph sampling methodologies, including the sampling technique used in practice. In summary, in this work we present: (1) The first extensive study of graph-based sampling prior to training for inductive recommender systems; (2) A holistic evaluation of the limitations of current sampling methodologies and inductive RSs; and (3) A set of interesting research directions for the design of sampling techniques in inductive recommender systems. **Our results demonstrate that: (i)** It is possible to maintain good predictive performance by training on 50% of the data while decreasing, in this way, the training time by up to 85%. **(ii)** Temporal sampling and user-based sampling perform best. **(iii)** For datasets with a high popularity bias, it is often enough to use 5% of data for the system to perform well; and **(iv)** with sampling ratios below 50% existing sampling techniques and existing RSs still struggle to maintain good performances; this raises the question of whether it is indeed possible to design more representative sampling algorithms and more robust learning approaches.

2 BACKGROUND AND PRELIMINARIES

Similar to previous studies [21], we consider RSs using users, items, and positive interactions as input data. Further, we also allow for textual information and attributes attached to items. Formally, given a set of users $\mathcal{U}$ and a set of items $\mathcal{I}$, we define an interaction matrix $\mathbf{I}\in\{0,1\}^{|\mathcal{U}|\times|\mathcal{I}|}$, where $\mathbf{I}_{ui}=1$ if a user $u\in\mathcal{U}$ has interacted with an item $i\in\mathcal{I}$; otherwise $\mathbf{I}_{ui}=0$, i.e., the user has never interacted with the item. The interaction information can be structured as a bipartite graph, known as a CG, where rating interactions appear as edges. Thus, the CG can be defined as a $\mathcal{G}_{cg}=\langle\mathcal{V}_{cg},\mathcal{R}_{cg}\rangle$, where $\mathcal{V}_{cg}=\mathcal{U}\cup\mathcal{I}$ are the users and items and $\mathcal{R}_{cg}=\{(u,i)|\mathbf{I}_{ui}=1\}$. Furthermore, we define the mapping function $\mathcal{T}_t:\mathcal{U}\times\mathcal{I}\rightarrow\mathbb{N}^{\geq 0}$, mapping all rating interactions (edges) to a natural number representing the time at which the rating was made, represented as t_i in Figure 1. The temporal aspect of ratings are important as trends and user interests change over time. The evaluation of RSs should, therefore, take temporal information into account when constructing train, validation, and test sets.

In addition, a Knowledge Graph (KG) [2, 21, 44], a heterogeneous graph containing entities and their semantic relations, is added to model descriptive information for items (see Figure 1). A KG is a directed labeled multigraph defined as the triple $\mathcal{G}_{kg}=\langle\mathcal{V}_{kg},\mathcal{R}_{kg},\mathcal{L}\rangle$ including nodes for both recommendable entities (items) and descriptive entities ($\mathcal{V}_{desc}=\mathcal{V}_{kg}\setminus\mathcal{I}$). Furthermore, the labels $\mathcal{L}$ represent the semantic type of edges, s.t. the relationship can be defined as $\mathcal{R}_{kg}\subseteq\mathcal{V}_{kg}\times\mathcal{L}\times\mathcal{V}_{kg}$. In this model, the KG does not represent the collaborative signal; thus we combine the KG and CG as a Collaborative Knowledge Graph (CKG) [21, 44], st, $\mathcal{G}_{ckg}=\langle\mathcal{V}_{ckg},\mathcal{R}_{ckg},\mathcal{L}_{ckg}\rangle$, where $\mathcal{V}_{ckg}=\mathcal{V}_{kg}\cup\mathcal{U}$, $\mathcal{R}_{ckg}=\mathcal{R}_{kg}\cup\{(u,likes,i)|\mathbf{I}_{ui}=1\}$, and $\mathcal{L}_{ckg}=\mathcal{L}\cup\{likes\}$. We further include a feature function $\mathcal{X}:\mathcal{V}_{kg}\rightarrow\mathbb{R}^d$ mapping each entity to a feature vector representing, for instance, the textual information for the node and the structure of $\mathcal{G}_{kg}$.

We treat the recommender objective as a ranking problem. Thus, a recommender is a function $\mathbf{r}_{u,\mathcal{I}}=\mathcal{F}_\theta(\mathcal{X},\mathbf{I},\mathcal{G}_{kg},u)$ parametrized by learned parameters θ producing a ranking score for all items in $\mathcal{I}$ according to the inferred preferences of user u. Thus, given

the ranking $\mathbf{r}_u$, it must hold that $\forall i, i' \in \mathcal{I}$, with $i \neq i'$, we have that $\mathbf{r}_{u,i} > \mathbf{r}_{u,i'}$ iff. the user u prefers item i over i'.

Given the above data model and a sampling ratio α, a sampling method $\mathcal{S}$ produces subgraphs $\mathcal{G}'_{cg} \sqsubset \mathcal{G}_{cg}$ and $\mathcal{G}'_{kg} \sqsubset \mathcal{G}_{kg}$ such that $|\mathcal{G}'_{cg}|+|\mathcal{G}'_{kg}| \leq \alpha \cdot (|\mathcal{G}_{cg}|+|\mathcal{G}_{kg}|)$. As in prior work [49], our goal is to train on the subgraphs $\mathcal{G}'_{cg}$ and $\mathcal{G}'_{kg}$ to learn the parameters for $\mathcal{F}_\theta$ and perform inference on the full graph.

3 RELATED WORK

In the inductive setting, we have users and items not seen during training, for which the RS should be able to make recommendations. This capability is crucial for real-world applications where users and items are continuously added. Further, it allows to train on a sub-graph while performing predictions for the entire graph.

Inductive Recommender Systems. There are multiple methods for inductive recommendation, using different techniques ranging from graph-based methods [21, 46] and transformer-based [35, 39], to RSs based on variational encoders [53]. However, a large pool of methods, as shown in Table 1, can make inductive recommendations for either only new users or only new items. Hence, a method able to recommend to new users would still need to train on the full set of items and vice versa. Meta-learning methods can recommend to new users and with new items bu shortly training on the new data [14, 25]. Numerous techniques propose using subgraphs based on user-item pairs, alleviating the need for learned user and item embeddings; instead, using the graph structure and distances to generate embeddings [50, 52]. However, constructing distinct subgraphs for each pair is prohibitively time-consuming and space-consuming when ranking items [21, 46]. Several methods use user meta-data to improve recommendations [4, 43], but such data is often unavailable or limited to a small user subset [38]. Privacy and data constraints limit interest in these methods.

Graph-based approaches use Graph Neural Networks (GNNs) to perform aggregation over all nodes in the graph. To reduce the training overhead, GraphSAGE [15] applies node sampling during batch constructions, fixing the memory overhead. GraphSAGE was designed for node classification and thus does not support recommendation lists. Among inductive recommender systems, INMO [46] instead learns initial embeddings for a subset of users and items, which all nodes must aggregate from for their representation. Thus, it does not use node features. Yet, for very large graphs, using only neighbor sampling was insufficient, and PinSAGE [49] thus applied both sampling of the training graph and introduced a MapReduce framework to scale-out the computation. Notably, PinSAGE is designed to recommend pins to boards, which can be translated to users and items; however, contrary to users, the boards are not explicitly modeled by PinSAGE and the method thus focuses on item-item recommendation exploiting in this way the collaborative signal. Instead of relying only on the collaborative signal, GInRec [21] proposes using KG information, applying relation-specific gates for aggregation, and simply representing users by their neighbors. When subsampling the graph, we naturally remove both users and items for which we are still interested in recommending. Methods unable to handle such scenarios are, therefore, not relevant. Consequently, the relevant recommenders that we can examine are PinSAGE [49], INMO [46], and GInRec [21].

Table 1: Related recommendation methods, the Task they support among (C) Node Classification, (R) Ranking, (P) Rating Prediction, and (SR) Sequential Recommendation.

Model	Task	Inductive User	Inductive Item	Architecture	Main Limitation
BERT4Rec [40]	SR	✔	✗	Transformer	Cannot recommend for new items
IDCF [45]	P	✔	✗	Matrix factorization	
ReBKC [20]	P	✔	✗	Multi-headed attention	
IGCCF [11]	R	✔	✗	GNN	
BSARec[39]	SR	✔	✗	Transformer	
ICP [51]	R	✗	✔	NN	Cannot recommend for new users
GAR [6]	R	✗	✔	Adversarial learning	
CVAR [53]	R	✗	✔	Variational encoder	
MeLU [25]	R	(✔)	(✔)	Meta-learning	Requires retraining for each new user
MetaKG [13]	R	(✔)	(✔)	Meta-learning	
IGMC [52]	P	✔	✔	Subgraph	User-item subgraph construction is cost-intensive
GIMC [50]	P	✔	✔	Subgraph	
PGD [43]	R	✔	✔	Student/teacher model	Requires user metadata
IHGNN [4]	R	✔	✔	GNN	
GraphSAGE [15]	C	(✔)	✔	GNN	Not made for recommendation
PinSAGE [49]	R	(✔)	✔	GNN w/ attention	
INMO [46]	R	✔	✔	GCN	
GInRec [21]	R	✔	✔	GNN w/ gates	

Training efficiency. Multiple approaches exist for reducing the graph sizes other than sampling[17]: (i) graph sparsification removes edges and/or nodes to reduce the computational cost while preserving performance. Using top-k nodes or edges has been used based on various scoring metrics, such as PageRank [34] or through a parameterized method [23]. (ii) Graph coarsening merges nodes into supernodes either through reconstruction or other optimization strategies [17]. The reconstruction can be either through spatial, by merging pairs with the least effect on the reconstruction error, or spectral methods, by comparing the eigenvalues or vectors. Alternatively, it is possible to learn supernodes, representing a cluster of the graph [19]. (iii) Graph condensation, constructs a synthetic graph for which a method can be trained on with similar performance [17]. They use gradient matching between a method learned on the original graph and the synthetic graph, distribution matching of the properties, or trajectory matching. Sparsification and condensation methods are usually designed for node classification and almost always rely on labeled nodes [17, 41]. Furthermore, the condensation methods can be both time and space intesive [41, 47].

Sampling of graphs has been used for approximate spectral clustering [42], for topology estimation [24], estimating graph characteristics [5], and covariance estimation [8]. However, none of these study node embedding methods. Many GNN methods apply sampling during training, requiring recomputation at each batch. They can be grouped largely into node-wise, layer-wise, and subgraph-based methods [31]. Nevertheless, the sampling is always performed on the *full train graph*, repeatedly, which can be infeasible in practice. Within hyperparameter optimization, multiple methods exist to decrease tuning time, with one branch focusing on dataset sampling [12, 32]. However, after finding optimal parameters, the methods still require training on the full graph due

to working with transductive methods. Other works use sampling to adaptively select negative sampling for faster training; however, they still require all positive samples [7]. Instead, PinSAGE [49] was shown to be able to train on a random uniform sampling over the graph. Specifically, sampling 20% of all graph boards that, for their dataset, proved to have negligible impact on performance. However, the final graph after sampling still contained multiple millions of nodes and their graph was not the usual bipartite or multi-partite graph. Thus, the question about which sampling technique is more effective and what are the actual effects on different dataset size and domains remains open. For example, random node sampling would sample sporadic nodes and create loosely connected graphs, which is less suitable for graph convolution methods.

Therefore, for the first time, we study different graph-based sampling techniques for state-of-the-art inductive methods. We choose to focus on well established sampling methods that ensure semi-coherent graph structures [26].

4 METHODOLOGY

We detail here the sampling methods and the inductive recommender systems used. For the recommenders, we describe only the most important parts contributing to their performance.

4.1 Sampling Methods

We evaluate two standard graph sampling approaches described as the most scalable and effective for reducing the size of very large graphs and designed specifically for their ability to preserve structural properties of the graphs [26]. The sampling technique adopted by PinSAGE [49], and a simple baseline taking into account the temporal information on edges. We perform node sampling for all methods, producing an induced subgraph where all connecting edges among the sampled nodes are preserved. When sampling from the KG, we limit the starting nodes to nodes for the CG.

Forest Fire (FF) [27]. FF simulates a tree burning process, where nodes ignite neighbors based on probabilities. It uses two edge probabilities: forward p_f and backward p_b. We test two edge-sampling methods: FFB with a binomial mean of $(1-p)^{-1}$ [27], and FF with mean np. The latter is greedier in selecting edges when encountering a hub, thus terminating earlier, but it produces very skewed distributions, as shown in the experimental section. Given a random starting node, the method ignites both backward and forward-going edges; the new burning nodes can now also burn their neighbors, and thus, the forest fire continues. If no new burning nodes exist, a new random start node is selected. We present the FF algorithm in Figure 2 to illustrate the use of the sampled input nodes.

Random Walk (RW) and Random Jump (RJ) [34]. RW randomly selects a starting node and performs random walks from it with a restart probability p_c; adding visited nodes to the frontier. If at each step of the walk no new nodes could be visited, a new node is picked as the starting node. RJ is a similar method that randomly jumps to a new node during the walk, with the same probability p_c.

PinSAGE Sampling (PS) [49]. When training PinSAGE [49], "board" sampling is proposed for training using a smaller graph. In this case, the graph is a bipartite graph between boards and pins, and when a board is sampled, itself and all its pins are added to the sample until some criteria is met. We adapt this sampling method by simply sampling users and their interactions for the CG. However, this only works for bipartite graphs, and adapting it to the KG is non-trivial. We use RW for KG sampling since a taxonomy path describes meaningful connections. Further development for heterogeneous graphs remains an open research question.

Temporal Sampling (TS). As each rating is associated with a time t, we can sample the users and items that have been active most recently. Meaning, given a CG, we sample the user u and item i, s.t. the rating time is newer than that of any other user u' and item i' rating, as $\mathcal{F}_t(u,i) \geq \mathcal{F}_t(u',i')$. Then, given that the KG does not have any timestamps, we use RW for that portion of the graph.

Time Complexity. The complexity of FF is $O(|\mathcal{V}| + |\mathcal{R}|)$ when sampling all edges as the algorithm when starting at the root in a tree structured graph, as the algorithm is equivalent to breath first search. For RW and RJ, the worst case would be a graph of disconnected nodes containing only self loops, $O(|\mathcal{V}|lw)$, where l is the walk length and w is the number of walks performed per node. PS goes through all users and their interactions, the complexity is thus $O(|\mathcal{U}| + |\mathcal{R}|)$. Finally, for TS, the complexity is $O(|\mathcal{R}|)$ as the method in the worst case need to visit each edge.

Algorithm 1 General sampling architecture

input: $\mathcal{G}_{cg}, \mathcal{G}_{kg}, \alpha$, Sampler
output: $\mathcal{G}'_{cg}, \mathcal{G}'_{kg}$, where $|\mathcal{R}_{cg}| \cdot \alpha \approx |\mathcal{R}'_{cg}| \wedge |\mathcal{R}_{kg}| \cdot \alpha \approx |\mathcal{R}'_{kg}|$
$\mathcal{G}'_{cg} \leftarrow$ Sampler$(\mathcal{G}_{cg}, \{\}, \alpha)$
$\mathcal{G}'_{kg} \leftarrow$ Sampler$(\mathcal{G}_{kg}, \mathcal{V}_{kg} \cap \mathcal{V}'_{cg})$

Algorithm 2 Forest Fire algorithm

Require: SampleNeighbors: samples neighbors of a node given probabilities and NodeSubgraph: constructs a subgraph containing only input nodes and the edges of the resulting graph.

function ForestFire$(\mathcal{G}, \mathcal{V}_{in}, \alpha, p_f, p_b)$
  $e \leftarrow |\mathcal{G}| \cdot \alpha$ ▷ Number of edges to sample
  ▷ Initialize burning, frontier, and #samples
  $B \leftarrow \{\}, F \leftarrow \{\}, s \leftarrow 0$
  $w \leftarrow$ Random start node from (F **if** $F \neq \emptyset$ **else** $\mathcal{V}$)
  while $s \leq e \wedge s \leq |\mathcal{G}|$ **do**
    $N \leftarrow$ SampleNeighbors$(\mathcal{G}, w, p_f, p_b)$
    $B \leftarrow B \cup N \cup \{w\}$
    $F \leftarrow F \cup \{w\}$
    if $\mathcal{V}_{in} \setminus F = \emptyset$ **then**
      $S \leftarrow \mathcal{V} \setminus F$ **if** $B \setminus F = \emptyset$ **else** $B \setminus F$
      $w \leftarrow$ Random node from S
    else
      $w \leftarrow$ Random node from $(\mathcal{V}_{in} \cup B) \setminus F$
    $s \leftarrow |$NodeSubgraph$(\mathcal{G}, B)|$
  return NodeSubgraph$(\mathcal{G}, B)$

4.2 Inductive Recommenders

The best-performing methods for recommendation in this setting are all based on GNNs and perform graph convolutions. A GNN can be described using an aggregation function and an update function, the former computing a neighborhood representation of nodes and the latter updating the node [15]. For example, the neighborhood aggregation can be the mean of its neighborhood, followed by a non-linear layer as an update function [15]:

$$\mathbf{e}_{\mathcal{N}_v}^{(l)} = \frac{1}{|\mathcal{N}_v|} \sum_{(v',v) \in \mathcal{N}_v} \mathbf{e}_{v'}^{(l-1)}, \mathbf{e}_v^{(l)} = \sigma\left(\mathbf{W}\left[\mathbf{e}_v^{(l-1)} \| \mathbf{e}_{\mathcal{N}_v}^{(l)}\right]\right), \quad (1)$$

Table 2: Dataset properties: I: items; U: users; R: ratings; TR: ratings in test set; STime: is the skewness of the rating times using the Fisher-Pearson coefficient; and DCG/DKG are the densities for the Collaborative Graph and Knowledge Graph, respectively.

	#I	#U	#R	DCG	#TR	STime	#Entities	#Relations	#Relationships	DKG
MovieLens	4,645	14,206	1,889,382	2.86E-02	499,040	0.26	14,062	8	100,719	2.88E-04
Amazon Book	24,841	70,679	843,228	4.80E-04	322,048	-0.90	88,572	39	2,555,995	1.99E-04
Yelp	77,319	174,840	2,428,509	1.80E-04	809,989	-0.50	75,199	12	1,643,792	7.07E-05

where $l \in [1, \ldots, L]$ is the current layer, $\mathcal{N}_v$ is the neighborhood of $v \in \mathcal{V}$, $\mathbf{e}_v^{(l)} \in \mathbb{R}^d$ is the embedding at layer l, σ is some activation function, $\mathbf{W} \in \mathbb{R}^{d' \times d}$ is a linear layer, and $[.\|.]$ is concatenation. The initial embeddings of $\mathbf{e}_v^{(0)}$ can thus be represented either by using a learned embedding or by extracting features. We refer to the initial embedding of all vertices as $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d^0}$. A GNN captures information from distant nodes through multiple graph convolutions. Each convolution acts like a bounded BFS, so graph size directly impacts training time.

Pin SAmpling and aggreGatE (PinSAGE [15]). PinSAGE's node embeddings are created, based on the GraphSAGE architecture, using the feature function $\mathcal{X}$, allowing the method to perform inductive recommendation. PinSAGE was used to recommend pins to boards, i.e., user collections of similar items. Thus, the method only uses an item graph $\mathcal{G}_i$, where edges represent the co-pinning (co-interactions) of the items, thus not representing the users. For example in Figure 1, "The Prestige" and "Don Jon" would be connected as Aiden likes both of them. Hence, PinSAGE optimizes towards item similarity as:

$$\sum_{(i,i') \in \mathcal{G}_i} \mathbb{E}_{i'' \sim Pr(i)} \max(0, \mathbf{e}_i \mathbf{e}_{i''} - \mathbf{e}_i \mathbf{e}_{i'} + \Delta), \tag{2}$$

where $\Pr(i)$ is a probability of selecting a negative item given i.

Gated Inductive Recommender (GInRec [21]). GInRec proposes using KG information in addition to user interactions. The method uses relation-specific gates to capture the relational information. GInRec further applies an auto-encoder architecture over all input features given by $\mathcal{X}$ to reduce their dimensionality. In contrast to PinSAGE, users are represented in the graph using the CKG and initialized using a zero-vector, assuming graph convolutions are sufficient for user representations. The method can thus utilize Bayesian Personalized Ranking (BPR) loss for ranking [37], trying to rank positive items higher than negative items, which is optimized in conjunction with the auto-encoder loss.

Inductive Module for Collaborative Filtering (INMO [46]). INMO uses a key-query architecture, selecting a subset of nodes as keys, learning their representations and how to infer representations for non-key elements. Hence, it defines a subset $\mathcal{U}_k \subseteq \mathcal{U}'$ and $\mathcal{I}_k \subseteq \mathcal{I}'$, that can be used to represent all users and items as:

$$\mathbf{e}_u^{(0)} = \frac{1}{(|\mathcal{I}_u \cap \mathcal{I}'| + 1)^\alpha} \sum_{i \in \mathcal{I}_u \cap \mathcal{I}_k} \mathbf{e}_i^{(-1)} + \mathbf{e}_{user}, \tag{3}$$

$$\mathbf{e}_i^{(0)} = \frac{1}{(|\mathcal{U}_i \cap \mathcal{U}'| + 1)^\alpha} \sum_{u \in \mathcal{U}_i \cap \mathcal{U}_k} \mathbf{e}_u^{(-1)} + \mathbf{e}_{item}, \tag{4}$$

where $\mathbf{e}_i^{(-1)} \in \mathbb{R}^{d^{-1}}$ are the learned embeddings, e_{user} is a learned template embedding and $\mathcal{I}_u = \{i | (i,u) \in \mathcal{N}_u\}$. For item queries, the same equations are used, although they are inverted. Since all vertices in the CG are represented as the average embedding, the individuality of the learned embeddings is lost. Therefore, in tandem with the BPR loss, INMO proposes a self-enhancing loss:

$$\sum_{u \in \mathcal{U}'} \sum_{i \in \mathcal{I}_u \cap \mathcal{I}'} \sum_{i' \in \mathcal{I}' \setminus \mathcal{I}_u} ln\, \sigma \left(\mathbf{e}_u^{(-1)\top} \mathbf{W}_s \mathbf{e}_i^{(-1)} - \mathbf{e}_u^{(-1)\top} \mathbf{W}_s \mathbf{e}_{i'}^{(-1)} \right) \tag{5}$$

While user $\mathbf{e}_u^{(0)}$ and item $\mathbf{e}_i^{(0)}$ embeddings can be used by any subsequent recommender, INMO adopted LightGCN [18].

5 EXPERIMENTS

As in PinSAGE [49], we aim at reducing the amount of resources needed and the computation cost of training by using a subsampled graph, while inference is still performed on the full graph. Extended results can be found in [22]. We answer the following questions: **RQ1)** How do sampling methods affect the ability of RSs to learn reliable models? **RQ2)** How does sample size affect the models' performance? **RQ3)** What is the correlation between training time and performance when sampling? **RQ4)** How do different RSs models handle subsampling?

Datasets. We evaluate the methods on three real-world datasets (See Table 2): (i) a dataset with ratings on movies, MovieLens-20m (ML-20m) [16] extended with the MindReader KG [3]; (ii) one with reviews of books, Amazon-Book (2014) (AB) [33] for which a KG was constructed when testing the transductive method KGAT [44]; and (iii) a dataset with reviews of businesses, Yelp Dataset (YD) [48], for which we extracted a KG [9]. For each dataset, we only use ratings for items connected to the respective KGs, removing all other items and the respective ratings. We further remove users and items with less than 5 ratings as well as users with ratings spanning less than 5 days. The datasets are split with ratios 0.8:0.1:0.1 for train, validation, and testing, respectively; ensuring that all ratings of the train set occur before the validation set, and validation before test. This ensures trends occur naturally over time and that all methods are tested on new data. The sampling is performed on the training partition, as we are interested in reducing the training time. We sample a few ratings for each user for validation and testing, simulating new users being greeted with an initial page where they provide initial ratings, similarly to [21, 25].

Analyzing the datasets (Table 2), we notice that the ratings are unevenly distributed over time. We report their skewness in the STime column. Positive values mean most ratings occur early, with less activity later; negative values indicate the inverse. We observe that most of the ratings for AB occur late, while most occur early for ML-20m. Such distributions naturally affect the subsequent results, where AB and YD probably adhere to normal business growth. However, the YD was affected by COVID, as seen in Figure 2, tracking the number of ratings given to an item per month, with a moving window of 12 months. Making the dataset non-trivial for RSs.

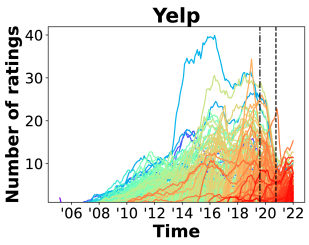

Figure 2: #Ratings on items.

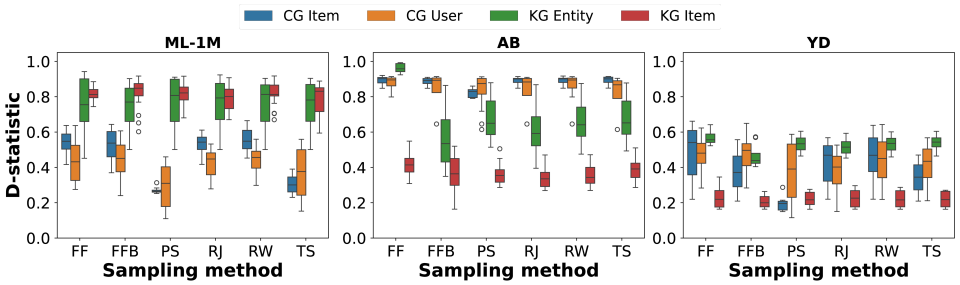

Figure 3: Kolmogorov-Smirnov D-statistic of degree distribution per node type.

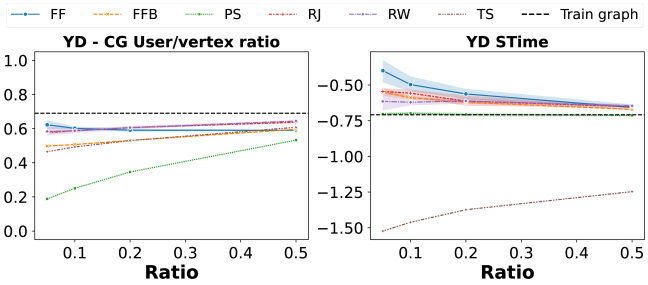

Figure 4: Illustration of user ratio and rating skew in the YD.

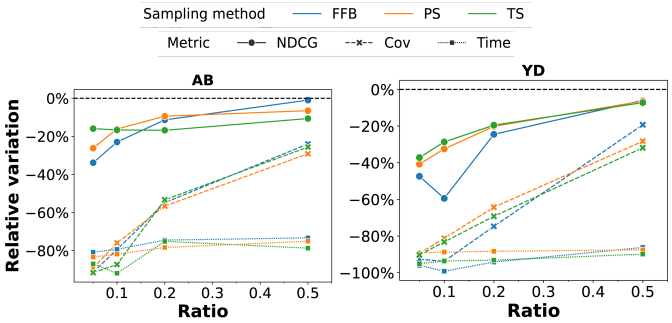

Figure 5: INMO's relative performance.

Table 3: Sampling resource usage with sampling ratio $r = 0.5$. The sampling time (Time) for CG/KG is in seconds, peak memory (Mem) in GB, and MTT is the floored average method train time. TC is time complexity with l and w being walk length and number of walks, respectively.

	ML			AB			YD			
	Time		Mem	Time		Mem	Time		Mem	TC
	CG	KG		CG	KG		CG	KG		
FF	<1s	5s	0.9GB	46s	<1s	1.2GB	192s	80s	1.5GB	$\lvert\mathcal{V}\rvert + \lvert\mathcal{R}\rvert$
FFB	17s	15s	0.9GB	290s	504s	1.2GB	1,272s	1,108s	1.5GB	$\lvert\mathcal{V}\rvert + \lvert\mathcal{R}\rvert$
PS	2s	-	1.2GB	12s	-	1.2GB	66s	-	1.5GB	$\lvert\mathcal{U}\rvert + \lvert\mathcal{R}\rvert$
RJ	8s	10s	0.9GB	96s	194s	1.3GB	442s	298s	1.6GB	$\lvert\mathcal{V}\rvert l w$
RW	8s	11s	0.8GB	58s	100s	1.2GB	152s	109s	1.5GB	$\lvert\mathcal{V}\rvert l w$
TS	4s	-	0.9GB	22s	-	1.3GB	74s	-	1.4GB	$\lvert\mathcal{R}\rvert$
MTT	3,860s	-	-	15,670s	-	-	77,728s	-	-	

Parameters. For the samplers, we study different ratios $\alpha \in \{0.05, 0.1, 0.2, 0.5, 1\}$, going from an extreme sub-sampling setting to the full graph. Due to space constraints, full details for 0.05 and 0.1 are reported in the extended version on the online repository, while their implications are still discussed below. As in related works [26, 27], we set the forward probability at $p_f = 0.35$, the backward probability at $p_b = 0.2$, the jump/restart probability at $p_c = 0.15$, and set the walk length at 10. We have implemented all methods in PyTorch, testing each method's implementation until we achieved a similar performance on the original datasets reported. For parameter tuning, we apply Asynchronous Successive Halving (ASHA) [28], a method based on multi-armed bandit methodology of high initial exploration of parameter combinations, before focusing on fewer combinations. Methods are tuned on the full graphs, using the same hyperparameters in all sampling settings. We tried tuning on $\alpha \in \{0.2, 0.5\}$ using PS getting similar performance as tuning on the full graph on these ratios. PS has been used in the industry, we thus want to validate its performance in other settings [49]. All configurations are available on the online repository with additional experimental results in [22]. We use an NVIDIA A10 GPU, dual processor setup with Intel Xeon Gold 6326, and 256 GB RAM.

Features. For feature extraction, we utilize the average textual embeddings of Sentence-BERT [35] for all items, with the texts being the first paragraph of the Wikipedia page, when available, otherwise Wikidata, for ML-20m and AB, and review text for YD. Furthermore, we compute the node degrees normalized by centering around zero and scaling to unit variance. The scaling is calculated for the train features and applied to the validation and test features. However, the descriptions of entities can be non-descriptive (see https://www.wikidata.org/wiki/Q20656232). We, therefore, use TransR [29] to generate embeddings for all descriptive entities.

Evaluation metrics. We rank all items in the test set, as ranking a subset has been shown to skew the results [36]. We exclude items already interacted with since multiple ratings between the same user and item cannot occur [44]. We use four standard ranking measures: NDCG@k, recall@k, precision@k, and PR-AUC, and one serendipity measure, coverage@k [1]; reporting the average performance over all users. A high coverage is not indicative of the recommendation performance, as a RS giving random recommendations would have high coverage but low ranking ability; therefore, it cannot be looked at in isolation. However, for brevity, *we only report HR and NDCG* in the comparison table Table 4, as the other results confirm the same findings we report here.

Sampling viability. We report the sampling time and maximal memory usage during a run (implemented in Python without any parallelization) for sampling ratio 0.5 in Table 3. We note that it is possible to greatly optimize the sampling process for even faster running times. Yet, we see that their running time is already orders of magnitude shorter than the respective training time, confirming the possible gain in runtime reduction when adopting sampling.

We see that FFB is often the method with the longest running time. That is because its sampling strategy selects only few nodes at every iteration, i.e., each step burns only one or two neighbors, meaning a higher number of iterations compared to a sampler constructing a densely connected neighborhood. To select more nodes we tried increasing the sampling probabilities s.t. the binomial mean would be around 10. With higher sampling likelihood, the sampling speed reduces to that of RJ. Furthermore, we see that the KG sampling of FF is also considerably faster than that of other methods for the AB dataset. This is due to presence of nodes with high degree. We analyzed the degree distribution and found a great skew in the degrees, as the $99.99th$ percentile has around $2k$ edges, while the $99.999th$ percentile has $100k$ edges in the AB KG. We note sampling large portions of edges of few hubs is not desirable leading to poor distribution similarities as seen for AB in Figure 3.

We use the Kolmogorov-Smirnov D-statistic over the *cumulative distribution function* of the in- and out-degrees for comparing the shapes of the distributions to evaluate the sampling methods ability to maintain representative structural properties of the original graph [26]. Lower values indicate a greater alignment between the sampled graph and the original graph. We sample for each sampling method and ratio combination five times, plotting the D-statistics for each node-type in Figure 3. Due to the heterogeneous nature of the KG, the non-uniform degree distributions, and the restriction to consider already samples entities, the sampling methods have more difficulty in approximating the structural properties of the KG. Nonetheless, it would not make sense to relax the restriction on the seeding of the sample for the KG, since information detached from the items would be irrelevant for item recommendation.

Current sampling methods treat edges and nodes as homogeneous. They ignore node types, likely causing distribution misalignment across types. Future work should develop methods tailored to bipartite CGs and heterogeneous graphs like KGs.

RQ1 & RQ2. The summary of the result in terms of NDCG and AUC compared to the reduction in training time is reported in Table 4. We observe similar trends also for the other metrics not reported here. **Choice of the best sampler depends on the dataset, ratio, and RS used.** Interestingly, while PS is able to better capture the CG's degree distribution (Figure 3), TS is the best performing of the two in most cases. Further, we notice that both **PS and TS perform well in many settings**, often being best or second best performing. For example, for the AB dataset, TS and PS are the best performing for all recommender methods for most sampling ratios. Yet, we see a trend where TS becomes the best performing for all datasets and methods when $\alpha \leq 0.1$. This indicates that **for small datasets data recency is important** compared to capturing correct distributions and that it is very important in sparse situations for CF.

As it would be expected, in most cases, reducing the amount of training data reduces the prediction quality of the models. This is most clearly seen on the YD, where all methods have trouble generalizing properly without the full datasets, regardless of the sampling method. When sampling 50% of the graph, the final prediction qualities across the sampling methods present only limited differences. For **high ratios the sampling method is less important** as sufficient users, items and ratings have been sampled regardless of sampling methodology. Yet, as previously stated, this is not true at lower ratios. This raises the question on the data-efficiency of these methods, i.e., whether these recommendation systems are actually able to infer inductive bias from complex graph structures or are just aggregators of collaborative signal (discussed in RQ4). When **sampling 5% to 10%, none of the methods can recommend well compared to their baseline performance on the whole graph**. The only exception is for the dense and popularity biased ML-20m dataset. However, all methods perform better than a naïve TopPop recommender [10] with only 5% of the data, except GInRec on YD. Only INMO is capable of decent recommendations when using 20% on the AB dataset and maintains performance with 50% of the data for the YD. We find **neither node-type distribution nor rating time similarity correlate with ranking performance**, as illustrated in Figure 4. For example, for YD with $r = 0.1$, we find that PS performs better than RW in all cases; however, RW's user ratio is far closer to the train graphs ratio than PS's. For rating distribution, there is little difference between TS and PS performance. Yet, PS almost perfectly matches the base graphs' rating skew while TS is far off. There appears to exist a **slight correlation between the degree distribution CG and the performance of the sampler** exists, as PS and TS are frequently best performing and have the lowest d-statistics. However, the absolute value is not indicative of performance and cannot be compared across datasets.

RQ3. Generally, reducing the amount of training data significantly reduces the training time. Yet, when using 50% of the data, PinSAGE's training time does not decrease at the same rate of other methods. This is likely due to the item-item loss function. For **PinSAGE and GInRec we see that longer running times often correspond to better ranking performance**. These methods jointly learn input feature representations, aggregation functions, and ranking composition. Therefore, learning to represent and aggregate node features may require more computation and data to learn inductive biases. In contrast, INMO only optimizes initial node embeddings for the ranking objective. The reduced training time is particularly important for INMO, which does not use node sampling, having exponential training time w.r.t. the number of edges (see Figure 5). INMO has a time complexity of $O(L|\mathcal{R}|d)$ for propagation. This is required for each edge in the train graph, leading to an epoch complexity of $O(L|\mathcal{R}|^2 d)$, where L is the number of layers and d the dimensionality.

On YD, we observe a 4% decrease in performance and 86% decrease in training time. Although it seems like a substantial decline, **a 4% decrease may effectively be negligible in practice**. As the HR for INMO is 0.17 and decreases to 0.163 when r=0.5, this means that in a ranked list of 20 items, the method would show at least one relevant item in both settings (on average). Even when INMO's HR performance is reduced by 30% for r=0.05, it still recommends at least one relevant item in the top 20, with a HR of 0.119. Furthermore, the sampling maintains the ordering of methods in

Table 4: Results of methods at different sampling ratios and with different methods. All results are measures at $k = 20$, except AUC which evaluates the complete list, with running time in hours. Bold indicates the best performing within a group.

		GInRec									INMO									PinSAGE								
		ML-1M			AB			YD			ML-1M			AB			YD			ML-1M			AB			YD		
		HR	NDCG	Time	HR	NDCG	Time	HR	NDCG	Time	HR	NDCG	Time	HR	NDCG	Time	HR	NDCG	Time	HR	NDCG	Time	HR	NDCG	Time	HR	NDCG	Time
- -		**0.734**	**0.202**	**0.9**	**0.132**	**0.029**	**2.7**	**0.122**	**0.021**	**7.9**	**0.663**	**0.185**	**1.2**	**0.154**	**0.038**	**7.3**	**0.171**	**0.033**	**45.5**	**0.583**	**0.152**	**1.1**	**0.122**	**0.027**	**3.0**	**0.136**	**0.024**	**11.3**
0.20	FF	-5.1%	-10.3%	-75.5%	-34.2%	-33.9%	-82.6%	-62.5%	-68.0%	**-84.3%**	-5.8%	-12.9%	**-87.0%**	-13.3%	-19.3%	-75.5%	-20.5%	-26.5%	-90.0%	-1.5%	-15.3%	**-87.7%**	-26.6%	-33.7%	-85.3%	-41.7%	-50.7%	-89.4%
	FFB	-9.5%	-12.6%	-78.0%	-31.0%	-30.1%	-79.5%	-58.2%	-64.0%	-61.6%	-3.6%	-6.6%	-85.2%	-9.1%	-11.3%	-74.5%	-18.8%	-24.5%	**-94.2%**	**11.3%**	**-0.0%**	-59.4%	-27.3%	-29.4%	-83.6%	-30.2%	-34.5%	-64.7%
	PS	-6.9%	-8.7%	-68.4%	-38.8%	-29.3%	-89.1%	-61.5%	-68.9%	-82.5%	**-1.4%**	**-1.1%**	-84.5%	-6.9%	-9.4%	**-78.4%**	**-15.5%**	-20.2%	-88.3%	-3.9%	-3.4%	-51.7%	-25.5%	-31.6%	-87.3%	-35.5%	-42.3%	-81.6%
	RJ	-10.0%	-15.7%	**-83.3%**	-35.8%	-37.6%	-81.6%	-64.5%	-70.8%	-73.0%	-3.7%	-8.6%	-86.8%	-6.2%	-6.3%	-73.2%	-22.2%	-27.3%	-92.6%	-4.9%	-4.0%	-74.0%	-17.8%	-26.5%	-82.0%	-41.1%	-47.2%	**-89.6%**
	RW	-12.3%	-16.4%	-81.1%	-35.6%	-27.5%	**-89.9%**	-60.3%	-66.6%	-82.1%	-3.6%	-7.1%	-84.7%	**-3.8%**	**-5.3%**	-75.9%	-18.3%	-24.4%	-90.7%	8.2%	-0.6%	-79.1%	-45.5%	-56.7%	**-92.5%**	-34.6%	-41.9%	-69.8%
	TS	**-4.1%**	**-6.6%**	-69.7%	**-17.1%**	**-2.9%**	-85.0%	**-48.5%**	**-55.0%**	-58.5%	-7.0%	-7.6%	-86.0%	-14.2%	-16.8%	-75.1%	-15.5%	**-19.5%**	-93.2%	6.7%	-0.9%	-59.3%	**-15.5%**	**-21.0%**	-86.1%	**-20.5%**	**-25.3%**	-73.0%
	*NS	-42.8%	-71.7%	-	-68.6%	-69.4%	-	-92.2%	-94.2%	-	-3.8%	-7.0%	-	-39.2%	-43.1%	-	-24.3%	-31.5%	-	1.8%	-32.7%	-	-46.0%	-44.6%	-	-82.1%	-86.5%	-
0.50	FF	-7.7%	-10.5%	-60.2%	-28.4%	-29.5%	-77.8%	-46.5%	-52.6%	-67.9%	-0.4%	-3.0%	-67.0%	-1.8%	-0.6%	-69.7%	-6.6%	-10.0%	-89.5%	-3.3%	-3.2%	**-55.0%**	-20.7%	-30.7%	**-74.0%**	-18.3%	-21.8%	-41.3%
	FFB	-6.7%	-8.9%	**-72.5%**	-29.1%	-29.1%	**-82.5%**	-47.3%	-53.1%	-63.7%	**-0.4%**	**-0.4%**	-70.9%	-2.1%	-1.0%	-73.3%	**-4.4%**	-6.0%	-86.2%	9.2%	6.1%	2.9%	-18.4%	-19.7%	-55.6%	**-14.8%**	**-17.3%**	-6.1%
	PS	-5.2%	**-4.9%**	-50.8%	-30.5%	-30.7%	-66.5%	-49.9%	-57.7%	**-73.9%**	-0.8%	-0.4%	-70.8%	-5.6%	-6.5%	-75.1%	-4.6%	-6.3%	-87.5%	**13.2%**	**6.4%**	-14.3%	-10.2%	-13.8%	-4.0%	-16.6%	-19.3%	-16.3%
	RJ	**-5.1%**	-5.9%	-17.4%	-26.4%	-26.1%	-63.1%	**-42.3%**	**-50.2%**	-48.5%	-0.6%	-2.0%	-65.8%	**-0.4%**	**1.2%**	**-80.1%**	-5.2%	-7.1%	-85.4%	4.8%	2.0%	-26.1%	-9.1%	-14.0%	-49.4%	-15.5%	-18.6%	-18.5%
	RW	-8.5%	-16.6%	-55.4%	**-19.6%**	**-15.7%**	-61.2%	-47.9%	-55.4%	-66.9%	-1.5%	-3.2%	-63.6%	-3.7%	-0.8%	-64.7%	-4.6%	**-5.8%**	-85.1%	-0.1%	0.4%	-6.4%	-12.5%	-15.9%	-50.9%	-16.7%	-20.4%	-21.2%
	TS	-5.3%	-6.1%	-55.1%	-27.1%	-25.6%	-80.7%	-42.9%	-50.6%	-57.3%	-5.1%	-5.6%	**-73.7%**	-10.4%	-10.6%	-78.7%	-5.8%	-7.2%	**-89.8%**	7.5%	2.8%	-34.7%	**-8.3%**	**-11.2%**	-65.8%	-19.1%	-23.8%	**-70.1%**
	*NS	-18.0%	-42.0%	-	-48.8%	-47.5%	-	-78.3%	-84.3%	-	-1.8%	-4.0%	-	-19.1%	-20.1%	-	-8.5%	-11.9%	-	-21.0%	-50.5%	-	-41.6%	-46.0%	-	-70.1%	-75.1%	-

* Defined in RQ4.

Table 5: Summary of recommendations and insights for RQ1–RQ4

RQ1: Samplers	RQ2: Sampling ratio	RQ3: Training time	RQ4: Recommenders
TS/PS perform best in most settings. For ratios ≤0.10 use TS. Samplers matching the degree distribution perform better.	r=0.5 maintains recommendation performance with great time savings. $r \leq 0.10$ does not maintain performance on datasets with less than 2.5mil ratings unless popularity biased.	80% time reduction at $r = 0.5$ for GNN-based recommender without neighborhood sampling. Attention requires longer time for apt performance. Subsampling ≥0.20 for hyperparameter tuning is possible.	Learned features are superior in most settings. When side information or KG features are key to performance, ensure enough data is retained to cover relevant entities. Methods learning collaborative bias are less dependent on the samplers.

almost all cases. Thus, it is **generally possible to compare models on the sample** to infer their relative performance on the full graph [12, 32]. In almost all cases, the methods performance with $r \in \{0.05, 0.10\}$ was better than a naïve TopPop [10] recommender, only GInRec performing worse on YD. **INMO obtained up to 5x the performance for NDCG on** 0.05 **compared to TopPop.**

Finally, we find RSs give **less diverse recommendations when trained with less data**. In Figure 5, we highlight how INMO's coverage increases as more data is given to the method (similar trends are exhibited by the other baselines). More training data allow the method methods' to make diverse recommendations A small subset of items may be sufficient for the ML-20m dataset, but has high impact on the performance for other datasets. Future research should **ensure RSs are capable of providing diverse recommendations even when little data is available**.

RQ4. INMO seems to be the most robust method to downsampling. Interestingly, while GInRec uses external knowledge, this seems to reduce the effectiveness of the method when there is little information available. **The current methods cannot effectively exploit the KGs' signal**. Interestingly, PinSAGE performs best on ML-20m, likely due to ML-20m being popularity biased and as the textual attributes of movies are of far higher quality than what is available for both AB and YD. However, all methods struggle on the YD. The dataset is likely more difficult compared to others as a dramatic change in most popular items occurs due to COVID, as seen in Figure 2. To investigate whether methods prioritize collaborative signal or structural information, we design a NichéSampler (NS) sampling items with the fewest user ratings. For methods that learn inductive bias, the features present in the graph used for training dictates which inductive biases are learned by the model. Thus, with a training graph with only niche items, if the methods maintains the same predictive performance, it is because they rely only on collaborative signal, disregarding learned rules. Looking at Table 4, we observe that all methods show a performance drop under NS, but GInRec and PinSAGE see a larger drop in performance. Therefore, **all methods learn some form of inductive bias**. Yet, INMO likely relies more on the collaborative signal, which is expected as it does not have any attention mechanisms on edges. This means that **collaborative filtering can be learned from small samples**, while more robust inductive biases from graph structures requires different sampling techniques and learning architectures. TS and PS perform well at lower ratios, suggesting they capture meaningful signals. Thus, leveraging time and representative user selection offers a promising path for improved sampling.

6 CONCLUSION AND FUTURE WORK

We investigate the practical implications of employing subsampled training data with different graph-based sampling methodologies when training inductive RSs. Inductive techniques are able to provide predictions for out-of-samples data and past evaluations have exploited this ability to reduce training time. Our evaluation shows that the PinSAGE and Temporal sampling approach produces the most reliable samples. Yet, for all RSs, at least 50% of the training graph is required to maintain prediction accuracy except for popularity biased datasets where ≤ 10% is sufficient. Nonetheless, the most robust method, INMO, showcases an important reduction in training time, with a decrease of over 80%, already with 50% of the graph. Future research could design sampling methods for and increase the robustness of inductive recommendation methods.

ACKNOWLEDGMENTS

This research was partially funded by the Independent Research Fund Denmark (DFF) under grant agreement no. DFF-8048-00051B and the Poul Due Jensen Fond (Grundfos Foundation).

REFERENCES

[1] Gediminas Adomavicius and YoungOk Kwon. 2012. Improving Aggregate Recommendation Diversity Using Ranking-Based Techniques. *TKDE'12* (2012), 896–911.

[2] Russa Biswas, Lucie-Aimée Kaffee, Michael Cochez, Stefania Dumbrava, Theis E. Jendal, Matteo Lissandrini, Vanessa López, Eneldo Loza Mencía, Heiko Paulheim, Harald Sack, Edlira Vakaj, and Gerard de Melo. 2023. Knowledge Graph Embeddings: Open Challenges and Opportunities. *TGDK* 1, 1 (2023), 4:1–4:32.

[3] Anders H. Brams, Anders L. Jakobsen, Theis E. Jendal, Matteo Lissandrini, Peter Dolog, and Katja Hose. 2020. MindReader: Recommendation over Knowledge Graph Entities with Explicit User Ratings. In *CIKM'20*. Association for Computing Machinery, 2975–2982.

[4] Desheng Cai, Shengsheng Qian, Quan Fang, Jun Hu, and Changsheng Xu. 2023. User Cold-Start Recommendation via Inductive Heterogeneous Neural Network. *TOIS'23* (2023).

[5] Emrah Çem, Mehmet Engin Tozal, and Kamil Saraç. 2013. Impact of sampling design in estimation of graph characteristics. In *IPCCC'13*. 1–10.

[6] Hao Chen, Zefan Wang, Feiran Huang, Xiao Huang, Yue Xu, Yishi Lin, Peng He, and Zhoujun Li. 2022. Generative Adversarial Framework for Cold-Start Item Recommendation. In *SIGIR'22*.

[7] Xiaohui Chen, Jiankai Sun, Taiqing Wang, Ruocheng Guo, Li-Ping Liu, and Aonan Zhang. 2023. Graph-Based Model-Agnostic Data Subsampling for Recommendation Systems. In *KDD'23*. 3865–3876.

[8] Sundeep Prabhakar Chepuri and Geert Leus. 2017. Graph Sampling for Covariance Estimation. *TSIPN'17* (2017), 451–466.

[9] Mads Corfixen, Magnus Olesen, Thomas Heede, and Christian Filip Pinderup Nielsen. 2023. The Yelp Collaborative Knowledge Graph. https://doi.org/10.5281/zenodo.8049832

[10] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys'10*. 39–46.

[11] Edoardo D'Amico, Khalil Muhammad, Elias Z. Tragos, Barry Smyth, Neil Hurley, and Aonghus Lawlor. 2023. Item Graph Convolution Collaborative Filtering for Inductive Recommendations. In *ECIR'23*.

[12] Noemí DeCastro-García, Ángel Luis Muñoz Castañeda, David Escudero García, and Miguel V. Carriegos. 2019. Effect of the Sampling of a Dataset in the Hyperparameter Optimization Phase over the Efficiency of a Machine Learning Algorithm. *Complex.* 2019 (2019), 6278908:1–6278908:16. https://doi.org/10.1155/2019/6278908

[13] Yuntao Du, Xinjun Zhu, Lu Chen, Ziquan Fang, and Yunjun Gao. 2023. MetaKG: Meta-Learning on Knowledge Graph for Cold-Start Recommendation. *TKDE'23* (2023).

[14] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *ICML'17*.

[15] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS'17*. 1025–1035.

[16] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *TIIS'15* (2015), 1–19.

[17] Mohammad Hashemi, Shengbo Gong, Juntong Ni, Wenqi Fan, B. Aditya Prakash, and Wei Jin. 2024. A Comprehensive Survey on Graph Reduction: Sparsification, Coarsening, and Condensation. In *IJCAI'24*. 8058–8066.

[18] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *SIGIR'20*. 639–648.

[19] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. 2021. Scaling Up Graph Neural Networks Via Graph Coarsening. In *KDD'21*, Feida Zhu, Beng Chin Ooi, and Chunyan Miao (Eds.). 675–684.

[20] Bei Hui, Lizong Zhang, Xue Zhou, Xiao Wen, and Yuhui Nian. 2022. Personalized recommendation system based on knowledge embedding and historical behavior. *Appl. Intell.'22* (2022).

[21] Theis E. Jendal, Matteo Lissandrini, Peter Dolog, and Katja Hose. 2023. GInRec: A Gated Architecture for Inductive Recommendation using Knowledge Graphs. In *KaRS'23 (CEUR Workshop Proceedings)*. 80–89.

[22] Theis E. Jendal, Matteo Lissandrini, Peter Dolog, and Katja Hose. 2025. The Limits of Graph Samplers for Training Inductive Recommender Systems: Extended results. arXiv:2505.14241 [cs.IR]

[23] Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. 2022. Graph Condensation for Graph Neural Networks. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net. https://openreview.net/forum?id=WLEx3Jo4QaB

[24] Maciej Kurant, Minas Gjoka, Yan Wang, Zack W. Almquist, Carter T. Butts, and Athina Markopoulou. 2012. Coarse-grained topology estimation via graph sampling. In *WOSN'12*. 25–30.

[25] Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. 2019. Melu: Meta-learned user preference estimator for cold-start recommendation. In *SIGKDD'19*.

[26] Jure Leskovec and Christos Faloutsos. 2006. Sampling from large graphs. In *SIGKDD'06*. ACM, 631–636.

[27] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *SIGKDD'05*. ACM, 177–187.

[28] Liam Li, Kevin G. Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. 2020. A System for Massively Parallel Hyperparameter Tuning. In *MLSys'20*. mlsys.org.

[29] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *AAAI'15*.

[30] David C. Liu, Stephanie Kaye Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C. Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. 2017. Related Pins at Pinterest: The Evolution of a Real-World Recommender System. In *WWW'17 Companion*. 583–592.

[31] Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, and Dongrui Fan. 2022. Sampling Methods for Efficient Training of Graph Convolutional Networks: A Survey. *IEEE CAA J. Autom. Sinica* 9, 2 (2022), 205–234. https://doi.org/10.1109/JAS.2021.1004311

[32] Matteo Montanari, Cesare Bernardis, and Paolo Cremonesi. 2022. On the impact of data sampling on hyper-parameter optimisation of recommendation algorithms. In *SAC '22: The 37th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, April 25 - 29, 2022*, Jiman Hong, Miroslav Bures, Juw Won Park, and Tomás Cerný (Eds.). ACM, 1399–1402. https://doi.org/10.1145/3477314.3507158

[33] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. In *EMNLP-IJCNLP'19*. Association for Computational Linguistics, 188–197.

[34] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report. Stanford InfoLab.

[35] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP-IJCNLP'19*. 3982–3992.

[36] Steffen Rendle. 2019. Evaluation metrics for item recommendation under sampling. *arXiv preprint arXiv:1912.02263* (2019).

[37] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI'09*. 452–461.

[38] Juan Manuel Rodriguez and Antonela Tommasel. 2024. Leveraging User History with Transformers for News Clicking: The DArgk Approach. In *RecSysChallenge'24*.

[39] Yehjin Shin, Jeongwhan Choi, Hyowon Wi, and Noseong Park. 2024. An Attentive Inductive Bias for Sequential Recommendation beyond the Self-Attention. In *AAAI'24*.

[40] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. In *CIKM'19*.

[41] Qingyun Sun, Ziying Chen, Beining Yang, Cheng Ji, Xingcheng Fu, Sheng Zhou, Hao Peng, Jianxin Li, and Philip S. Yu. 2024. GC-Bench: An Open and Unified Benchmark for Graph Condensation. *CoRR* (2024).

[42] Nicolas Tremblay and Andreas Loukas. 2020. *Approximating Spectral Clustering via Sampling: A Review*. 129–183.

[43] Shuai Wang, Kun Zhang, Le Wu, Haiping Ma, Richang Hong, and Meng Wang. 2021. Privileged Graph Distillation for Cold Start Recommendation. In *SIGIR'21*.

[44] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. Kgat: Knowledge graph attention network for recommendation. In *SIGKDD'19*. 950–958.

[45] Qitian Wu, Hengrui Zhang, Xiaofeng Gao, Junchi Yan, and Hongyuan Zha. 2021. Towards Open-World Recommendation: An Inductive Model-based Collaborative Filtering Approach. In *ICML'21*.

[46] Yunfan Wu, Qi Cao, Huawei Shen, Shuchang Tao, and Xueqi Cheng. 2022. INMO: A Model-Agnostic and Scalable Module for Inductive Collaborative Filtering. In *SIGIR'22*. 91–101.

[47] Zhenbang Xiao, Shunyu Liu, Yu Wang, Tongya Zheng, and Mingli Song. 2024. Disentangled Condensation for Large-scale Graphs. *CoRR* (2024).

[48] Yelp. 2025. Yelp Open Dataset. https://www.yelp.com/dataset/. Accessed: 2025-05-20.

[49] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD'18*. 974–983.

[50] Chengkun Zhang, Hongxu Chen, Sixiao Zhang, Guandong Xu, and Junbin Gao. 2022. Geometric Inductive Matrix Completion: A Hyperbolic Approach with Unified Message Passing. In *WSDM'22*.

[51] Chuxu Zhang, Huaxiu Yao, Lu Yu, Chao Huang, Dongjin Song, Haifeng Chen, Meng Jiang, and Nitesh V Chawla. 2021. Inductive Contextual Relation Learning for Personalization. *TOIS'21* (2021).

[52] Muhan Zhang and Yixin Chen. 2019. Inductive Matrix Completion Based on Graph Neural Networks. In *ICLR'19*.

[53] Xu Zhao, Yi Ren, Ying Du, Shenzheng Zhang, and Nian Wang. 2022. Improving Item Cold-start Recommendation via Model-agnostic Conditional Variational Autoencoder. In *SIGIR'22*.