Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Crumbling Third-Party Cookies: a Recipe to Personalized Content Recommendation with Federated Graph Learning

Emilien Duc

Master's Thesis – September 2, 2024.
Institute of Information Security,
Secure & Trustworthy Systems Group.

Supervisors: Prof. Dr. Shweta Shinde
Prof. Dr. Prateek Saxena
Aashish Kolluri

Abstract

Designing effective recommendation frameworks for personalized user-item interactions in decentralized settings is a critical challenge in a climate of rising privacy concerns. For instance, Google's privacy sandbox initiative until recently has advocated for phasing out third-party cookies that enable centralized data tracking and aggregation with solutions such as FLoC, the Protected audience API, and the Topics API. Existing solutions suffer from various limitations, including poor accuracy compared to centralized approaches, trivial privacy leaks, and high communication and computation costs. In addition, we identify a critical vulnerability in existing privacy-preserving recommender systems, where an attacker can easily leak user-item relationships through a passive intersection attack during training.

In this work, we propose GRAPHITEE, the first federated graph learning-based recommendation framework that simultaneously addresses privacy, utility, and low resource requirements. GRAPHITEE leverages systems-level and algorithmic optimizations to enable privacy-preserving graph learning on fully decentralized user-item bipartite graphs while remaining resilient to our novel attack. GRAPHITEE achieves comparable utility to state-of-the-art centralized and decentralized baselines on a real-world news recommendation dataset. Moreover, GRAPHITEE 's low computational, bandwidth, and storage requirements makes it a viable solution to deploy on user devices, including mobile.

Contents

1 Introduction

Designing frameworks for serving personalized recommendations on user-item bipartite graphs in decentralized setups is an important problem in the world of rising privacy concerns. Such frameworks are motivated by the decentralization of private data, i. e., giving control back to users. For instance, Google's privacy sandbox initiative until recently has advocated for scrapping third-party cookies that enable centralized data tracking and aggregation. Instead, they have proposed solutions like FLoC, Protected Audience API, and Topics API. Their preferred solution, Topics API, reveals one topic the user is interested in in each visited website to allow interest-based recommendations. Alternatively, other frameworks have been proposed to address this issue by training machine learning (ML) models in the federated setup to perform local recommendations.

However, all of the frameworks today suffer from at least one of the following issues: bad accuracy compared to the centralized recommendation frameworks, trivially leaking the privacy of the users, and impractical communication and computation costs for the user. For instance, the Topics API, which has been deployed for testing by Google, has been shown to allow re-identification across sites for some users and does not provide useful recommendations [1][3][13]. However, it is very efficient and easy to integrate for publishers.

On the other hand, state-of-the-art ML-based recommendation frameworks are trained using a combination of secure aggregation and federated optimization algorithms and used locally to perform recommendations. Their designs are motivated to provide optimal utility, privacy, and communication costs. However, we show that they trivially leak users' private data by conducting a passive intersection attack during training, which the centralized aggregator can do. Conventional wisdom from a large body of research says that utility, communication costs, and privacy are usually at odds with each other. This highlights why it is very challenging to design such a framework and why, to the best of our knowledge, no solution that satisfies all three constraints reasonably has been proposed yet.

[1] Google has very recently canceled their deprecation of third-party-cookies [5].

1.1 Contribution

In this work, we propose the first recommendation framework, called GRAPHITEE, that addresses these three constraints of utility, communication costs, and privacy effectively. Overall, we claim the following contributions:

Conceptual:

- GRAPHITEE is the first federated graph learning-based framework that can be trained on fully decentralized user-item bipartite graphs, with no entity knowing any more raw data than their locally available data. We leverage multiple known and new systems-level and algorithmic optimizations to address utility, privacy, and communication constraints simultaneously.
- Unlike existing solutions, GRAPHITEE does not easily leak the raw data of individual users under intersection attacks and allows for implementing differential privacy if strict inferential privacy guarantees are needed.

Technical:

- We design a general intersection attack against many privacy-preserving frameworks that train machine learning models on user-item bipartite graphs. The attack can easily leak user-item relationships, and the vulnerability arises from updating item representations in every round during training.
- We carefully navigate through many possible design options and choose the following components:
  - Design a new GNN architecture for training, as GNN models are small and a natural fit for graph data, thus addressing communication and utility constraints.
  - Train the GNN using Retexo, a layerwise training strategy, to address communication and privacy constraints. Crucially, Retexo's training strategy helps significantly by not updating item representations in every training round.
  - Use enclave-based aggregation at the server side to address privacy and communication constraints. The enclave complements Retexo by enabling efficient graph learning, even when users do not know which other users have interacted with the same items. This makes our system resilient to the attack we design and precludes the clients from communicating with each other.

Evaluation:

- Utility is similar to the best centralized and state-of-the-art decentralized frameworks on the popular real-world news recommendation dataset called MIND. Furthermore, GRAPHITEE converges almost as fast as the state-of-the-art baselines.
- GRAPHITEE can be trained with half the communication costs incurred on clients compared to current baselines. The low computational and storage requirements make it a viable solution to be deployed on any user device, including mobile.

1.2 Organization of this Thesis

This thesis is structured as follows:

Chapter 2 formalizes our problem statement and provides a thorough overview of the background and related work on the various topics discussed in this work. Chapter 3 presents a novel intersection attack against bipartite recommender systems that update item representations in each round. We conclude the chapter with an overview of our innovative solution GRAPHITEE, which is not subject to the attack. In Chapter 4, we detail the design of GRAPHITEE step-by-step and formally define the protocols used for training and inference. In Chapter 5, we highlight the results of our experiments and analyze the implications of scaling the system to a fully distributed setup. We compare our results with existing recommender systems and highlight the limitations of Topics API for news recommendation. Finally, we conclude this thesis in Chapter 6[2].

[2]All the code related to this work can be found here: github.com/kisp-nus/GRAPHITEE

2 Problem Setup & Background

This chapter outlines the problem we aim to solve and provides essential background information on the various areas involved in this work. We start by defining the recommendation problem. We then present the constraints and challenges of a federated learning setup and our threat model. Then, we introduce essential background information on the fields we explore in this work and the relevant related work. Finally, we present a summary of the current challenges and the limitations of existing implementations.

2.1 The Recommendation Problem

Recommender systems infer users' preferences from user-item interactions or static features and further recommend items that users might be interested in. As the volume of digital content continues to grow exponentially, users face the challenge of information overload. Recommender systems aim to alleviate this problem by presenting users with a curated selection of items that align with their interests and preferences. Formally, consider a bipartite graph $\mathcal{G} : (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ are the set of nodes and edges, respectively. Nodes can be users ($u \in U$) or items ($i \in I$). We define an edge $e_{u \to i}$ between nodes u and i as a *click* or interaction from the user with the item. Given $\mathcal{G}$ and an unconnected user-item pair (u, i), the task is to predict whether u will interact with i or not. This is an edge-prediction task.

2.1.1 News Recommendations

Our specific focus in this work is on news article recommendations in a news feed. This problem is more complex because it requires us to extract information from the news articles themselves in addition to the user-item interactions. While the task is similar, this setting is especially relevant to our goal of developing a recommender system for real-time recommendations based on previous browsing behavior.

2.2 Federated Learning Setup

Introduced by Google in 2016 [11], Federated Learning (FL) has been an active area of research ever since. FL is a decentralized approach to machine learning in which participants (hereafter referred to as clients) collaboratively train a model without sharing their raw local data. As such, each client has their own local model, trained with their own data, but at every round, the clients update their model by taking an average of all the models' gradients. Multi-party secure aggregation for machine learning, as suggested by K. A. Bonawitz et al., is a common method to compute the aggregation of values without revealing any information on the private values themselves [4]. It requires participants to compute a mask with every other participant before sending out the masked gradients to be aggregated together.

In our scenario, each client has a local sub-graph $\mathcal{G}_l$ that includes the local user nodes and all the items and entities the users interacted with. User nodes are always unique, following the principle that we do not want the raw users' data to leave the device. Formally, $\forall u \in U, \exists! l$ such that $u \in G_l$. We use a user's history to create bi-directional edges to item and entity nodes interacted with by the user. Items and entity nodes may be replicated across partitions. There is no entity with a view of the complete graph $\mathcal{G}$. Nonetheless, we assume the presence of a controller that will coordinate the training.

2.3 Threat Model

Our federated training setup has two actors: the controller and the clients.

- *controller*: We consider an honest but curious controller. This means the controller will follow the training process correctly while extracting as much information as possible from the observed traffic. Naturally, the controller is aware of all clients involved but has no information on user interactions with items.
- *clients*: in the scope of this thesis focused on privacy, we consider the users to behave honestly but curiously.

2.4 Background & Related Work

2.4.1 Topics API

The Topics API categorizes users into broad interest groups based on browsing history. Each website (domain) a user visits is mapped to a topic using pre-computed

categorization or a deterministic model for unpopular websites. Every week, the browser computes the user's top 5 topics of interest based on their history. When a user visits a website, the browser gives out a random topic among the user's top 5 that the website can use for interest-based ads and content. Specific design details were added over time, such as serving completely random topics with a 5% probability for plausible deniability or prioritizing topics relevant to serve ads. Without third-party cookies, we believe this challenge of balancing personalization with privacy extends beyond advertising to other domains, such as news recommendations.

Other major browser vendors like Mozilla and Apple rejected the Topics API specification on the grounds that it provides little information to advertisers while severely risking user privacy and browsing safety [12][14]. After several delays and reports indicating an unsatisfactory performance of their privacy sandbox, Google recently canceled its plans to remove third-party cookies and wants to leave the choice up to users [5]. This change highlights the need for innovative, privacy-friendly solutions.

Multiple independent analyses of Topics API have been made since its first announcement in 2022. On the side of privacy, recent studies have criticized the Topics API, claiming that it allows re-identification and fingerprinting of users [3][1]. Regarding utility, the ad publisher Criterio reported an experiment on the Topics API that resulted in a drop of 60 % in their revenues, showing a limited utility of the Topics API [13].

Our work addresses both utility and privacy. The Topics API's struggles to achieve good utility demonstrate the need to leverage more user data when recommending items: if we want to preserve privacy, we need to perform the recommendations locally.

2.4.2 Graph Neural Networks for recommendations

Graph Neural Networks (GNNs) have been long used in recommendation tasks for their ability to capture high-level user-item interactions [20]. This comes from the fact that most information in recommender systems has a graph structure. GNNs are based on the concept of message passing. Nodes in the graph iteratively exchange information with their neighbors, updating their representations based on the aggregated messages. Having one message-passing layer in our model means that nodes will exchange information with their neighbor once. This process allows GNNs to capture local and global graph structures, making them powerful tools for relational data tasks. Different kinds of message-passing layers exist; in this work, we focus on SAGE.

In a homogeneous graph, a SAGE convolution layer to update the representation $\mathbf{h}$ of node v is simply defined as

$$\mathbf{h}_v^{(i+1)} = \text{SAGE}(\mathbf{h}_v^i, \mathcal{N}(v)) = \sigma \mathbf{W}^i \cdot (\mathbf{h}_v^i, \text{AGGR}(\{\mathbf{h}_u^i, \forall u \in \mathcal{N}(v)\})) \tag{2.1}$$

Where σ is the non-linearity, $\mathbf{W}$ is a matrix of trainable weights, $\mathcal{N}(v)$ is the neighborhood of v, and AGGR any aggregator such as *mean*, *sum*, or *pooling*[7].

In the specific context of news recommendation, GNNs have shown particular promise in modeling complex relationships between users, news articles, and contextual information. Taking into account additional information, such as user and news attributes, is key to relevant recommendations and requires more advanced GNN implementations. For instance, Ying et al. proposed PinSAGE, a recommender now used in production at Pinterest, that extracted an item-item graph based on the interactions between users and item features [22]. However, this model requires centralized training at its core. Zhang et al. recently presented DivHGNN, a heterogenous GNN with innovative modules that favor diverse recommendations while keeping the basics of graph learning: a small model based on message-passing[23]. Our work builds upon these advancements, incorporating some elements of DivHGNN's implementation while adapting them to a federated learning context to enhance privacy preservation and scalability.

2.4.3 Federated and Private News Recommenders

Most of the efforts on private news recommenders have been made in collaboration with Microsoft Research, the publishers of the MIND News Recommendation Dataset [19]. FedGNN is a well-known model to train a recommender in a federated setup [18]. The authors make use of homomorphic encryption and a third-party trusted server. However, their results have limited scalability and are evaluated on different datasets without making use of items' attributes. Fed4Rec and Efficient-FedRec are private recommenders that have shown to be very efficient on MIND by using language models to interpret the news articles' content in addition to the user-item interactions [21][15]. Both suffer from having a large model, which makes it harder and bandwidth-consuming to train in a federated setup. While Efficient-FedRec attempts to address this problem using a small user-side model and a large server-side model, they rely on repeated k-anonymity and user-to-user communication. In this work, we demonstrate how an intersection attack can cancel the privacy guarantees of k-anonymity and reveal the user's private data. We explore GNNs as an alternative that uses significantly smaller models and a novel training strategy called Retexo that cancels the need for repeated k-anonymity [9]. Our use of enclaves for secure aggregation reduces the bandwidth requirements and keeps users isolated from one another.

In the past, Ru et al. have explored the use of enclaves for secure aggregation when training a GNN and showed promising results. Nonetheless, their setup assumed

participants training on separate private datasets, whereas we assume a *shared* global graph. This key difference significantly increases the complexity of the task, as participants train on incomplete subgraphs without ever learning about the presence of other nodes.

2.5 Challenges

In summary, the challenges of designing a federated and privacy-oriented news recommendation system can be summarized as follows:

- **Privacy of user-item interactions**: users should not share their private interactions. Efficient-FedRec fails to achieve this by updating item representations in every round, which requires this information from users. Naïve GNN adaptations to the federated setup would also require users to give out this information in every round.
- **User-User anonymity**: users should not be aware of other users using the system nor required to communicate with them. Multi-party computation used by Efficient-FedRec fails to respect this principle. Moreover, this constraint is a significant challenge for graph neural networks as well because users are nodes in the graph themselves.
- **Computation efficiency**: a recommender system destined to be trained in fully distributed setups cannot rely on large models. Users have limited resources.
- **Communication costs**: the bandwidth usage should be limited and not prohibitive to the users. Multi-party computation is bandwidth-heavy and requires $O(N^2)$ exchanges every training iteration. Similarly, GNNs require communication with every neighbor at each iteration in order to perform the aggregation step of the convolution. Our use of an alternative training strategy called Retexo bypasses this requirement.
- **Utility**: most importantly, a recommender system should be able to make good recommendations. While already of limited utility for advertising, we show that Topics API could not be used for more advanced recommendation tasks. Similarly, FedGNN fails to leverage user and item attributes in the training process, struggling to gain further insights from the content of the items.

The different features and limitations of the mentioned related work and our solution GRAPHITEE are summarized in Table 2.1

Limitations of existing recommenders					
	User-Item	User-User	Computation	Bandwidth	Utility
Efficient-FedRec	✗	✗	✓	✗	✓
DivHGNN	✗	✗	✓	✗	✓
Topics API	✓	✓	✓	✓	✗
FedGNN	✓	✓	✗	✗	✗
GRAPHITEE	✓	✓	✓	✓	✓

Table 2.1: Comparison of the features and limitations of each different recommender system.

3 Our Approach

This chapter highlights a current vulnerability of private recommender systems that update item representations every training round. First, the vulnerability is defined, followed by an analysis of its efficiency on an existing state-of-the-art federated recommender. We conclude by introducing our innovative framework that does not suffer from this vulnerability.

3.1 Attack on Existing Bi-Partite Graph Recommenders

Updating item representations according to the interactions by the user is a common method used in different models, particularly GNNs, but also in Large Language Model (LLM) fine-tuning as does Efficient-FedRec. To address scalability concerns, federated learning models usually sample a small group of users every round to train and update the model [8]. Using some variant of secure aggregation, users will usually compute the aggregated gradient and the list of interacted items together. If the group is of size k, this process provides k-anonymity on the user-item interactions for the users in the group. The intuition behind the vulnerability is that if a user is sampled more than once, we can perform an intersection of both lists to reveal the user's individual interactions. The intersection process is formally described in algorithm 3.1. The first time a user is sampled, its anonymity group is all the users sampled in that round, and the candidate interactions are this round's items. During each following round where the user is sampled, the list of potential items a user has interacted with is reduced to the interaction between the previous and the current round's list. Similarly, the anonymity group is reduced to the intersection of all groups in which the user is sampled.

3.1.1 Attacking Efficient-FedRec

Applying our algorithm to Efficient-FedRec results in brutal re-identification of user-item interactions due to the small anonymity groups. On average, when running our attack during Efficient-FedRec's training, we could determine the individual interactions of 60.8% of all the users with absolute certainty. This is explained by a large number of users and a very small sampling size: if a user is sampled twice, it is extremely likely that no other user previously sampled with them is re-sampled at the same time. Given that the model needs to be trained for $2'100$

Algorithm 3.1: Intersection Attack

Data: Users, Items
Result: AnonymityGroups, Interactions

```
1  begin
2      AnonymityGroups ← Map(User, List[User]);
3      Interactions ← Map(User, List[Item]);
4      for r ← num_rounds do
5          k_users, items ← RoundInfo();
6          for u ← k_users do
7              if !∃AnonymityGroups[u] then
8                  AnonymityGroups[u] ← k_users;
9                  Interactions[u] ← items;
10             else
11                 AnonymityGroups[u] ← AnonymityGroups[u] ∩ k_users;
12                 Interactions[u] ← Interactions[u] ∩ items;
```

rounds and we sample 50 users each round, then the probability that a user will be sampled more than once is indeed $1 - P(\text{Not Sampled}) - P(\text{Sampled Once}) = 62.05\%$.

3.2 Overview of GRAPHITEE

We introduce GRAPHITEE, our framework designed to address the unique challenges of federated graph learning on decentralized user-item bipartite graphs. Our system leverages multiple known and new systems-level and algorithmic optimizations to address utility, privacy, and communication costs, while remaining elegantly immune to the attack described in section 3.1.

Our system's core is a Graph Neural Network made of 2 message-passing layers. The small nature of GNNs makes it a natural choice to reduce communication overhead while maintaining high utility. An obvious pitfall of this strategy for the federated setup is self-describing: message-passing. Trained in the standard way a GNN would be subject to the same attack we presented in section 3.1 and would also result in impractical communication costs [9]. For this reason, we employ the Retexo training strategy: this layerwise approach ensures that item representations are not updated in every round, a critical feature that protects from intersection attacks. Instead of having as many updates as there are rounds, we have as many updates as there are layers: 3. Given that we only update item representations 3 times, we can afford to have all users contribute to each update, removing the need for k-anonymity. Using Retexo further removes the communication between the clients in each training

round, which is typical for training a GNN in a distributed setup. This reduces the communication costs by several orders of magnitude over standard GNN training in the distributed setup [9].

Finally, we use a Trusted Execution Environment (TEE) for secure aggregation instead of multi-party computation to further enhance our system's privacy guarantees and scalability. This method provides confidentiality to users; it ensures that users remain anonymous to one another and the user-item relationships are not revealed to the controller. Using TEE reduces the time taken for aggregation and scales to thousands of users, unlike secure multi-party computation.

GRAPHITEE has been rigorously tested on the MIND dataset, a popular real-world news recommendation benchmark. The framework demonstrates utility on par with the best centralized and decentralized state-of-the-art solutions, with near-equivalent convergence speed. Importantly, GRAPHITEE achieves these results with significantly lower communication costs, making it suitable for deployment even on resource-constrained devices like mobile phones.

4 GRAPHITEE

In section 4.1, we describe our recommendation model and its different layers. In section 4.2, we present the federated setup in detail and define the training and inference protocols in this setting. This particularly includes our use of enclaves. Finally, in section 4.3, we make an analysis of the bandwidth usage of GRAPHITEE.

4.1 Our Model

DivHGNN, a recent model for diverse news recommendations, inspires our implementation [23]. Our model has three layers: a first representation layer to encode the nodes' attributes, followed by two message-passing layers. An overview of the model can be seen in Figure 4.1. We explain each part step-by-step in this section.

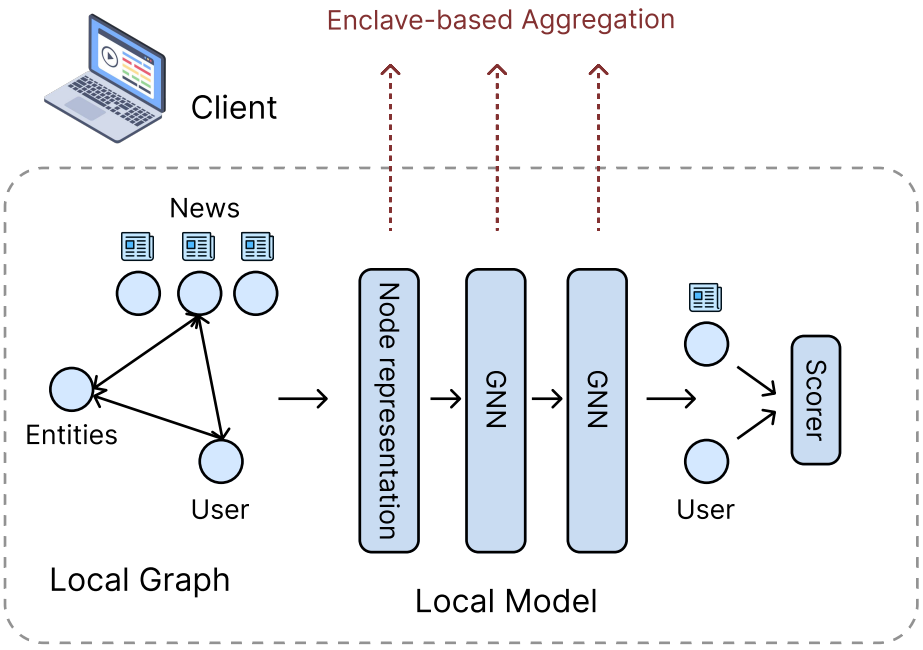

Figure 4.1: Different components of GRAPHITEE present on client devices.

4.1.1 Graph Creation

We create a heterogenous attributed graph in which each user and news article is a node. News article nodes have attributes for their category, title, and abstract. We encode those text-based attributes using a pre-trained language model. User attributes are simple statistics of their past interactions with news categories. In par with DivHGNN's implementation, we introduce a node type for "Entities". Entities are part of the dataset and represent different subjects, people, or items mentioned in the articles. They come with precomputed embeddings.

All edges in the graph are bi-directional. Intuitively, we define them as follows:

- *user*↔*news* is an edge if the user clicks on the news article.
- *user*↔*entity* is an edge if the user clicks on an article mentioning the entity.
- *news*↔*entity* is an edge if the entity is mentioned within the news article.

We build this graph using the users' historical interactions. In contrast with the graph used by DivHGNN, we do not have specific nodes for *words*. After noticing few to no improvements in the model's performance when using them, we decided not to include words as nodes to reduce the amount of stored and processed data.

4.1.2 Representation Layer

Accurate and expressive node representation is key to a high-performing model. We found that using IDs or a trainable embedding layer was insufficient to compete with recent recommendation models. This is why we use DivHGNN's node content adapter to align and fuse node representations. They demonstrated that this allows better performances and a more diverse set of recommendations[23]. Using such an adapter allows us to compete with a multi-head attention layer while having a model smaller by one to two orders of magnitude. The representation layer has 1.23 M parameters for a total size of 4.93 MB. It is composed of one linear layer per attribute type. The adapted attributes are fused together for each node type using a final attention layer to constitute the node's *representation*.

4.1.3 Graph Convolution Layers

Two graph convolutions follow the representation layer. Conceptually, this means that each node will perceive information from its 2-hop neighborhood. We implement a relation-based adaptation of GraphSAGE for a heterogeneous graph: each relation type r has its own SAGE^r module:

$$\mathbf{h}_v^{(i+1)} = \text{SAGE}^r(\mathbf{h}_v^i, \mathcal{N}^r(v)) \tag{4.1}$$

Messages from all relation types are then concatenated and crossed via self-attention. Crucially, the aggregation method we choose for SAGE is to take the *mean* of the neighboring nodes. This allows us to compute the aggregation only once per layer and use the layer-wise training method called Retexo[9]. We detail this process in subsection 4.2.1.

4.1.4 Edge-Prediction

After the message-passing, we apply a final linear layer to extract as much information as possible. The final representation is defined as:

$$h_{\{u,i\}}^{(L)} = \text{MLP}_{dense}(h_{\{u,i\}}^{(L-1)}) \tag{4.2}$$

Finally, we make an edge prediction using a *scorer*. For a user-item edge $e_{u \to i}$, we define the score as being the cross-entropy loss of both node representations $score(e_{u \to i}) = r_u \times r_i$. During training, we aim at minimizing the cross entropy loss between the label y and the computed score of each edge:

$$\text{CrossEntropy}(y,\ \text{Sigmoid}(\text{r}_\text{u} \times \text{r}_\text{i})) \tag{4.3}$$

4.2 Federated Training

We now understand the model trained and its components. This model and the local users' data are present on each client's device. In the following section, we highlight the process of federated training using a secure enclave and the different messages that are exchanged. The federated training using an enclave has four different steps:

- *initialization*: the controller sends out the model parameters to all clients. At this stage, the clients establish a symmetric key with the enclave that they will use throughout training. They can also check the enclave's attestation and fingerprint to ensure the right application is running.
- *local training*: each client trains on the model using their local data.
- *aggregation*: the clients encrypt their model's gradient using their symmetric key and send it to the enclave. The enclave aggregates all gradients and sends back the aggregated value. If strict inferential privacy guarantees are needed, the enclave can add noise to the aggregated gradient for differential privacy.

- *model update*: the clients update their local model using the received gradient. They can start local training again until convergence.

The particularity of our framework is that not only do clients train a *shared model*, but they also do so on a *shared graph*. In a naive implementation, all nodes must send and receive messages from their 1-hop neighborhood at each iteration. Recall that the item nodes are seen as belonging to the enclave; while we could maintain client-to-client privacy in this setting, the bandwidth usage would make it completely unscalable. We bypass this limitation using layer-wise training.

4.2.1 The Wisdom of Layer-Wise Training

We recall for convenience that a SAGE convolution layer is defined as

$$\text{SAGE}(\mathbf{h}_v^i, \mathcal{N}(v)) = \sigma \mathbf{W}^i \cdot (\mathbf{h}_v^i, \text{AGGR}(\{\mathbf{h}_u^i, \forall u \in \mathcal{N}(v)\})) \tag{4.4}$$

The elegant claim of Retexo is that we can train a convolution layer, meaning optimize the weights $\mathbf{W}$, without updating the aggregated representation from the 1-hop neighbors at each iteration. This means that instead of training the entire model together, we train one layer after the other. Once a layer is trained, we freeze the weights so that the representation of the nodes will not change when training the following layer. They demonstrated in their work that this could produce equivalent results to end-to-end training while reducing the bandwidth used by one to two orders of magnitude[9].

We adapt this idea to our setup: we only update the item's neighbor aggregation once after each layer. Each client needs to give the enclave their encrypted local representation, as well as their item interactions, such that the enclave can compute for each item

$$\mathbf{h}_{item} = \text{AGGR}(\{\mathbf{h}_{user}^i, \forall user \in \mathcal{N}(item)\}) \tag{4.5}$$

An astute reader might notice that this solution can potentially leak sensitive information about users. For example, if two users interacted with an item, this aggregation would simply leak the users' representations to one another. First, we can make sure not to update the item's aggregated neighborhood if less than k users interacted with it. In addition, we can add a custom amount of noise to each item depending on the number of users who interact with it.

4.2.2 Enclave-Based Secure Aggregation

Multi-party secure aggregation is a widely used method to compute the aggregation of values without revealing any information on the private values [4]. Unfortunately, this method has several downfalls for our use case. In addition to requiring 2 to 3 Round Trip Times (RTTs) per aggregation, each client needs to communicate with every other client in the process. This is unrealistic for our setting in which clients are users. Instead, we adopt a more efficient method of enclave-based aggregation.

The enclave has a known public key Pub_E that the clients use to verify the enclave's attestation and fingerprint to ensure the correct program is running. The complete protocol can be seen in algorithm 4.2. For each round, the enclave sends out the current model to the users; the users train locally, encrypt their gradient, and send it back to the enclave. Importantly, once a layer is trained, the enclave needs to update each item's neighborhood aggregation $\mathcal{N}$. This process happens only once per layer, that is 3 times for GRAPHITEE.

Algorithm 4.2: GRAPHITEE Distributed Training Protocol

Data: Clients, Enclave, model
Result: Trained GRAPHITEE model

```
1  begin
       /* Setup:  authenticate enclave and exchange symmetric key   */
2      1: Clients authenticate the Enclave;
3      2: Clients and Enclave exchange symmetric key;
       /* Layer-wise training                                       */
4      for each Layer do
5          for i ← 1 to num_iterations do
6              1: Enclave selects K random users and sends current model
                  parameters;
7              2: Users train locally;
8              3: Users send encrypted gradients to Enclave;
9              4: Enclave aggregates gradients and updates model;
           /* Update Item Nodes                                     */
10         1: Users send encrypted representations and interactions to Enclave;
11         2: Enclave updates item neighborhood aggregations;
12         3: Enclave sends all (interacted) items to each user (async);
```

4.2.2.1 Side-Channels

In this work, we do not restrict our framework to a specific enclave implementation such as SGX. Nonetheless, we assume that the enclave used will have typical

features of a TEE, such as isolation, attestations, and sealed storage, to protect from a malicious OS. Several enclave implementations have been found to contain multiple side channels over the years that can leak severe information on the data being processed. Side channels might be specific to each use case and implementation. For instance, if the application is not implemented in a time-safe fashion, it would be possible for the controller to infer how many items a client interacted with. Regarding communication, the messages exchanged can be of fixed size using padding such that no information is leaked from their size, as explained in section 4.3.

4.2.3 Inference

Once the model is trained, each client can have their final local model and perform recommendations locally. The attributes of the users are based on the user's history: as the user browses the web, their individual representation will be constantly evolving. Naturally, the neighborhood aggregation $\mathcal{N}$ of items remains fixed or can be updated at fixed intervals. When a user visits a website that wants to leverage the user's model, it will send out a list of multiple candidate items to recommend, along with their pre-computed neighborhood. The user adds those items to the local graph, performs a forward pass of the model, and finally scores the given edges. It can then display the items in the right order. This protocol is described in algorithm 4.3

Algorithm 4.3: GRAPHITEE Inference Protocol

Data: Client, Candidate items
Result: Local Recommendation
begin
`/* Recommendation task */`
1: Controller sends out a list of k items to a user (incl. neighborhood $\mathcal{N}$);
2: The user adds the item to its local graph and loads the last layer of the model;
3: The user computes the features of one forward pass on the graph;
4: The top-scoring items from the list are displayed in order to the user;

4.3 Communication

Let us analyze the communication cost of training GRAPHITEE. We consider entities and news articles under the same hat of *items* for simplicity and generalization. Recall we have n users equally split on N clients and a 3-layer model.

4.3.1 Model Updates

Updating the model is done using the model's gradient. Each layer's gradient size is 4.9, 3.5, and 3.5 MB. Each round requires a gradient aggregation. Therefore, the bandwidth used for gradient aggregation in a given round is given by Equation 4.6.

$$B_{grad} = \begin{cases} 4.9 \times \text{N MB} & \text{if Representation layer,} \\ 3.5 \times \text{N MB} & \text{if GNN layer.} \end{cases} \tag{4.6}$$

Nonetheless, as common in federated learning, we can limit the number of gradients to aggregate per epoch by only selecting random 1′000-5′000 clients to ask for their gradients [8]. In chapter 5, we make experiments with random batches of 2′000 users.

4.3.2 Message Passing

Message passing between nodes happens at the end of each layer. Users compute their local "user node" update locally, and the enclave needs to handle this part for the items. The enclave aggregates information from each item's neighboring nodes. As such, the users send their embeddings and the list of interacted items. A single-user embedding is of size 256×4 Bytes. Encoding the interactions in a one-hot vector would require 8 KB for our dataset. Therefore, the total bandwidth used for a round of item embedding updates is as follows:

$$B_{item\ update} = 8 \times \text{N} + 1 \times \text{n [KB]} \tag{4.7}$$

Once the enclave aggregates the neighborhood for each item from enough clients, it needs to send to each client the item embeddings they will need to train the next layer. This is the only size-sensitive exchange of information. If the enclave only sends the interacted items, the controller can know how many items the client has interacted with by observing the amount of data sent. We can mask this information using padding. In MIND News Recommendation Dataset, users interact on average with 20 items and at most with 500 items. This represents only 0.5 MB of data to send to each user.

5 Results

We have tested GRAPHITEE both centrally and deployed in a federated setup. The implementation leverages the Deep Graph Library (DGL) for graph learning and PyTorch Distributed for the coordination between the controller and clients [10][17]. We run our federated setup on AWS using low-resource `t3.medium` EC2 instances with only 2 CPUs and 4 GB of RAM for clients and an `m5.xlarge` instance with an inner `Nitro Enclave` for the controller and secure enclave [2]. We believe that the `t3.medium` instances reflect well the low resources available on a typical user device, including mobile phones.

This chapter addresses the following research questions:

- RQ1: What is the computation, time, and bandwidth cost of our federated system on users?
- RQ2: What is the overhead of using an enclave for secure aggregation?
- RQ3: How does GRAPHITEE performs in comparison with our News Recommender Systems baselines?
- RQ4: How does GRAPHITEE relate to Topics API when it comes to news recommendations?

We start by presenting the dataset's relevant statistics. We then analyze our distributed system using three cost factors: communication, time, and memory. Finally, we present our model's performance in comparison with well-known news recommendation systems.

5.1 Statistics of the Dataset

MIND News Recommendation Dataset is a widely used real-world dataset for news recommendation research. This dataset comprises user behavior logs from Microsoft News, collected over a period from October 12 to November 22, 2019. In our experiments, we utilize the small version of this dataset, which is made of $50'000$ random impressions. For MIND, the data is structured as follows: The news click history and records from the first four weeks make up the training set. The fifth week's records are used to create the validation sets. Finally, the test set is built using data from the last week of the collection period. MIND includes information on user interactions

with news articles, allowing researchers to analyze reading patterns and preferences. An *impression* is a sequence of articles shown to the users with labels indicating whether they were clicked on or not; this allows us to create positive and negative samples. The statistics are summarized in Table 5.1.

	# News	# Users	# Clicks	# Categories	# Sub-categories
MIND	65′238	94′057	347′727	18	270

Table 5.1: Statistics of the MIND News Recommendation Dataset.

Data partitioning is a crucial factor when training a distributed model. We performed a user-oriented partitioning that would represent the data present locally on a client's machine: we randomly split the users into N disjoint parts and replicated the items such that each partition contains all the items interacted with by its users. We uploaded all the pre-computed partitions to an S3 server so that each client can only retrieve its own data and build the local graph.

5.1.1 Baselines

Throughout this chapter, we compare our results to the following baselines, already introduced in chapter 2. The results here are based on measurements made on our own hardware using the same hyperparameters as those mentioned in the original work. We quickly summarize them again here:

- **DivHGNN** was our starting point when building GRAPHITEE and is a natural baseline [23]. In addition to the fact that this model can only be trained in a centralized setting, some key differences with our model include their use of Gated Attention for the GNN layers, an additional node type for *words*, and the use of topological information, which we removed as we assume an unknown global graph.
- **Efficient-FedRec** is a federated model presented for the very purpose of private news recommendation. It leverages a *small user-side model* and multi-party computation for gradient aggregation in order to fine-tune a Large-Language Model on the server side [21].

5.1.2 Metrics

Following previous work, we assessed the model using Area Under the Curve (AUC), Mean Reciprocal Rank (MRR), and n Discounted Cumulative Gain (nDCG) with n equal to 5 and 10 [21][6][15][23]. We briefly define and explain the intuition behind each of those.

- **AUC** is the area under the Receiver Operating Characteristic (ROC) curve, which plots the true positive rate against the false positive rate. It expresses well the trade-off between sensitivity (true positives) and specificity (false positives). An AUC of 1.0 indicates perfect prediction, while 0.5 is equivalent to random guessing.

- **MRR** focuses on the rank of the first relevant item in a list of recommendations. For news recommendations, it measures how high the first clicked article appears in the recommended list. MRR ranges from 0 to 1; the higher, the better.

- **nDCG** takes into account both the relevance and the position of recommended items. nDCG@5 and nDCG@10 specifically look at the top 5 and top 10 recommendations, respectively. Values range from 0 to 1, with 1 being the ideal ranking.

5.2 RQ1: User Cost of Distributed Training

This section explains several experiments to assess the impact of distributed training on the users.

5.2.1 Communication Cost

The dominant cost of communication for users will come from secure gradient aggregation. The size of the gradients is shown in Table 5.2. We can see that GRAPHITEE's gradients are significantly smaller than Efficient-FedRec. Our use of enclaves also significantly reduces the bandwidth requirements, as there is no need for multi-party computation.

Model Size and Aggregation			
Model	User Gradient Size	Server Gradient size	Aggr.
DivHGNN	-	3.1 MB	Centralized
Efficient-FedRec	8.72 MB	440 MB	MPC
GRAPHITEE	{4.93, 3.5, 3.5} MB	{4.93, 3.5, 3.5} MB	Enclave-based

Table 5.2: Comparison of the gradient sizes and aggregation methods for different models. The gradient sizes of GRAPHITEE are expressed per layer.

5.2.2 Time and Storage Cost

Training one round of the model can be split into the following phases: 1) forward pass, 2) compute the loss, 3) aggregate the gradient, and 4) backward pass. Intuitively, the time it takes to perform computational steps 1, 2, and 4 is dependent on the amount of data and the hardware available. We measured the total computational time of these steps when training either the representational or the GNN layer, depending on the number of items held locally by the clients. The averaged results over 500 rounds are shown in Figure 5.3. We can see that the computational time of the representation layer scales linearly with the data available, while the compute time for a GNN layer remains relatively low. This is explained by the simplicity and smaller size of the GNN layers.

Nonetheless, this experiment shows that for a user who interacted with less than 2′000 items, the overhead of GRAPHITEE on a low resource machine is minimal, with inferences in less than 50 ms. The storage cost remains similarly low and can be directly computed as being the size of the user's own representation and all the representations of the items they interacted with, of 1 KB each.

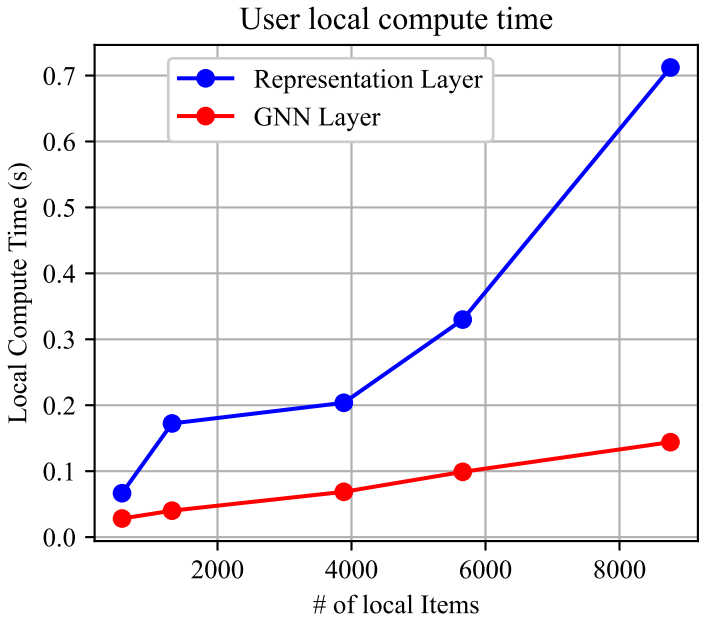

Figure 5.3: Computation cost incurred on the user depending on the number of items they interacted with for each layer type.

Comparison of Gradient Aggregation Techniques			
	Enclave	MPC	Controller-based
Exchanges	O(N)	O(N^2)	-
RTTs	1-2	2-3	1
Client-to-Client comm.	None	Required	None
Scalability (in N)	Linear	Quadratic	Linear
Differential Privacy Support	Global/Local	Local	Local
Outlier detection	Supported	Unsupported	Supported

Table 5.4: Comparison of the different gradient aggregation methods for federated learning with N clients.

5.3 RQ2: Use of Enclave-based Secure Aggregation

5.3.1 Comparison of Aggregation Methods

Naturally, securing the aggregation of gradients comes with a non-negligible overhead. However, compared to multi-party computation (MPC) aggregation methods used widely in federated learning today [4], we argue that enclave-based aggregation represents a lighter and more scalable solution. We compare both approaches and the centralized setup in Table 5.4. In all three cases, the clients will send the gradient to the controller, either in clear, encrypted, or masked. The key differences come from the preparation of the gradient: in order to encrypt them, the clients only need to establish a symmetric key with the enclave. When masking the gradients using MPC, the clients first exchange a mask with all other clients involved, resulting in N^2 exchanges. Client-to-client communication is a severe downside of MPC. Moreover, a previous study on recommendations using GNNs has found that adding differential-privacy noise on aggregated gradients has less impact on the utility of the model while preserving the same theoretical privacy guarantees of adding the noise locally [16]. This is only possible with enclave-based aggregation. Finally, while model poisoning from users is not part of our threat model, it is important to highlight that using MPC for aggregation prohibits the removal of individually malicious gradients, usually identified using outlier detection methods.

In our experiments, it took just 10ms for the `Nitro` enclave to decrypt the gradient of a user for the first layer. Thankfully, this process, as well as the aggregation of the decrypted gradients, can be parallelized, and the resources of the enclave can be increased, for example, by using more resources or multiple enclaves to process data in parallel.

5.3.2 Memory Requirements of the Enclave

Given the training and inferences protocols detailed in chapter 4, highlighting the memory requirements for the enclave is a straightforward endeavor. We recall that user and item representation are 1KB each. In a worst-case scenario, the enclave must remember all user representations and their interaction vectors, as well as all item representations and the ongoing computation of item representations for each GNN layer. This adds up to

$$M_{enclave} = n \times (1 + 8) + 2 \times 2 \times \|\mathrm{I}\| \times 1 \text{ KB} \simeq 1.05 \text{ GB on MIND} \tag{5.1}$$

5.4 RQ3: Performance Compared to News Recommendation Baselines

We now compare our graph model to known news recommendation baselines. The results displayed here have been computed centrally using 48 AMD EPIC 7443P 24-core processors, 256 GB of RAM, and an NVIDIA A40 GPU. We use an exponentially decaying learning rate to reduce the total training time and a sigmoid-activated binary cross-entropy loss. We apply a dropout rate of 0.2 and train each layer of our model for $7'000$ rounds. Our results are averaged over 5 runs and can be seen in Table 5.5. We discuss the metrics and baseline results below. GRAPHITEE-RB is a version of our model simulating the fully distributed setup by sampling batches of $2'000$ users every round for only $2'500$ rounds.

5.4.1 Performance Discussion

Overall, we can see that GRAPHITEE competes well with the state-of-the-art baselines. Given the multiple simplifications we made to the DivHGNN architecture and our lack of topological information, it is striking that we are able to achieve better results. Efficient-FedRec outperforms our model in terms of AUC by achieving an outstanding 67.6% compared to 66.9% and 66.3% for GRAPHITEE without and with batches, respectively. The highest performance of Efficient-FedRec is not surprising given that their server-side model is 2 orders of magnitudes bigger than GRAPHITEE. Nonetheless, we achieve almost equivalent MRR, nDCG@5, and nDCG@10, which demonstrates GRAPHITEE's ability to make relevant sequences of recommendations. This could be explained by our use of GNNs which captures well the interactions of the two-hop neighborhood, in addition to the design of our base model, which is focused on the diversity of recommendations. Performance decreases slightly when training with random batches. Multiple factors, including

MIND				
Model	AUC	MRR	nDCG5	nDCG10
DivHGNN	66.45	31.52	34.49	40.93
Efficient-FedRec	**67.63**	**32.48**	35.63	**41.90**
GRAPHITEE-RB	66.30	31.86	34.89	41.05
GRAPHITEE	66.87	32.34	**35.89**	41.77

Table 5.5: Performance comparison between GRAPHITEE and relevant baselines on the MIND News Recommendation Dataset. Results are expressed in percentages, and bold font is used to indicate the best overall performance.

the batch sizes and the number of epochs chosen, can explain this. However, the results remain within high-performing ranges.

5.4.2 Convergence

We consider that the model converges once it reaches 99% of its best performance within that run. Our model takes up to 2′000 rounds to converge in the centralized setup. Training using batches is known to extend the number of epochs required. We implemented a custom sampler that picks 2′000 training samples with replacements every round to simulate the distributed setup. This is a more representative approach than mini-batch sampling in which each data sample is sure to be used exactly once per epoch. We train a model that converges within 2′500 epochs when training with batches of 2′000 users. We show 2 plots comparing the validation AUC of Efficient-FedRec and GRAPHITEE in Figure 5.6. Overall, the plots show that both models converge in comparable epochs. We can see that each layer of GRAPHITEE contributes to increasing the performance. While the second GNN layer seems to add little utility when comparing AUC, we noticed that it significantly increased the performance of MRR, nDCG@5, and nDCG@10 (see Appendix A). This shows that the information from the 2-hop neighborhood, i. e the users interacting with the same items, contributes to a better sequence and ordering of recommendations.

Convergence and Training Time				
Model	Method	Batch-Size	Epochs	Total time
DivHGNN	Mini-Batches	128	24.78	4h24min
Efficient-FedRec	Random-Batches	50	2’100	1h26
GRAPHITEE-RB	Random-Batches	2’000	2’500	0h43min
GRAPHITEE	Full-Graph	n	2’000	0h14min

Table 5.7: Comparison of each model convergence and total training time with equal hardware resources and chosen batch sizes.

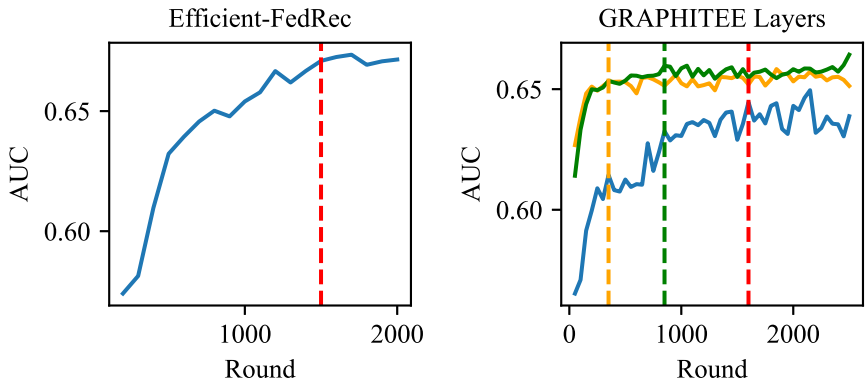

Figure 5.6: Evolution of validation AUC during training Efficient-FedRec (Left) and GRAPHITEE (right). The three layers of GRAPHITEE are blue, orange, and green, respectively. Dashed vertical lines indicate when 99% of the max performance is reached.

5.5 RQ4: GRAPHITEE and Topics API for News Recommendations

In this work, we explore a recommender system that aims to perform all recommendations on the local machine. This is in contrast to Topics API, which aims to enhance the performance of advertiser-based recommendations while leaking as little information as possible. Comparing the exact performance of the two systems would require us to develop a dedicated recommender system that takes a single user topic as input for each inference. Undoubtedly, such a model would perform far worse than a model with access to all previous user interactions, as confirmed by experiments made by leading ad publishers [13].

We further demonstrate the limitations of interest-based recommendations by simulating an adaptation of Topics API for our task. For each user, we compute their 5 most interacted topics by using the news article "subcategories". Because there are 270 different subcategories in the dataset, we believe it makes an adequate parallel with the 350 predefined Topics from the API. We only consider users who interacted with more than 5 different topics. We predict that a user is interested in a news article if its topic is in the user's top-5 and show the results in Table 5.8. Furthermore, we point out that the false positive and false negative rates are 18.78% and 64.97%, respectively. Those results emphasize that user interests, while relevant, are not enough for advanced recommendation tasks. Interests fail to capture the content of the news, the timing of the clicks, the order in which articles were read, and many more factors that can enhance recommendations. Moreover, while we used all 5 top user interests in our experiment, the Topics API is made to reveal only one topic at

a time, at random, further limiting its utility.

Topics API on MIND				
Model	AUC	MRR	nDCG5	nDCG10
Topics API	58.13	26.29	29.66	35.72
GRAPHITEE-RB	**66.30**	**31.86**	**34.89**	**41.05**

Table 5.8: Performance comparison between GRAPHITEE and an adaptation of Topics API. Results are expressed in percentages, and bold font is used to indicate the best overall performance.

6 Conclusion

This thesis presents GRAPHITEE, a novel decentralized recommendation framework that simultaneously addresses the key challenges of utility, privacy, and communication costs. Our implementation leverages Graph Neural Networks, the Retexo layer-wise training strategy, and enclave-based aggregation to provide strong privacy guarantees without compromising utility or incurring prohibitive communication costs.

Notably, GRAPHITEE's resilience to our newly designed intersection attack sets it apart from existing decentralized solutions, which we show trivially leak user data. Moreover, our smaller model and innovative training strategy reduce the bandwidth usage and hardware requirements of both clients and controllers while achieving performances on par with state-of-the-art recommenders.

This advancement demonstrates that GRAPHITEE is a viable solution to provide personalized recommendations in distributed setups. Consequently, we believe that it may offer an alternative solution to serve personalized advertisements if third-party cookies are deprecated in a climate of rising privacy concerns.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] M.S. Alvim, N. Fernandes, A. McIver, and G.H. Nunes: *The privacy-utility trade-off in the topics api*, 2024. https://arxiv.org/abs/2406.15309.

[2] I. Amazon Web Services: *Aws nitro enclaves*, 2024. https://aws.amazon.com/fr/ec2/nitro/nitro-enclaves/, Accessed: 2024-08-26.

[3] Y. Beugin and P. McDaniel: *A public and reproducible assessment of the topics api on real data*, 2024. https://arxiv.org/abs/2403.19577.

[4] K.A. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H.B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth: *Practical secure aggregation for federated learning on user-held data.* In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016. https://arxiv.org/abs/1611.04482.

[5] A. Chavez: *A new path for privacy sandbox on the web*, 2024. https://privacysandbox.com/intl/en_us/news/privacy-sandbox-update/, Accessed: 2024-08-26.

[6] S. Ge, C. Wu, F. Wu, T. Qi, and Y. Huang: *Graph enhanced representation learning for news recommendation.* In *Proceedings of The Web Conference 2020*, WWW '20, p. 2863–2869, New York, NY, USA, 2020. Association for Computing Machinery, ISBN 9781450370233. https://doi.org/10.1145/3366423.3380050.

[7] W.L. Hamilton, R. Ying, and J. Leskovec: *Inductive representation learning on large graphs*, 2018. https://arxiv.org/abs/1706.02216.

[8] P. Kairouz, H.B. McMahan, B. Avent, A. Bellet, M. Bennis, A.N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R.G.L. D'Oliveira, H. Eichner, S.E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P.B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konecný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, H. Qi, D. Ramage, R. Raskar, M. Raykova, D. Song, W. Song, S.U. Stich, Z. Sun, A.T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F.X. Yu, H. Yu, and S. Zhao: *Advances and open problems in federated learning.* Foundations and Trends® in Machine Learning, 14(1–2):1–210, 2021, ISSN 1935-8237. http://dx.doi.org/10.1561/2200000083.

[9] A. Kolluri, S. Choudhary, B. Hooi, and P. Saxena: *Scalable neural network training over distributed graphs*, 2024. https://arxiv.org/abs/2302.13053.

[10] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala: *Pytorch distributed: Experiences on accelerating data parallel training*, 2020. https://arxiv.org/abs/2006.15704.

[11] H.B. McMahan, E. Moore, D. Ramage, S. Hampson, and B.A. y Arcas: *Communication-efficient learning of deep networks from decentralized data*, 2016. https://arxiv.org/abs/1602.05629.

[12] Mozilla: *Mozilla standard positions: Topics api*, 2024. https://github.com/mozilla/standards-positions/issues/622, Accessed: 2024-08-26.

[13] T. Parsons: *Privacy sandbox testing results show shortfalls to meet cma requirements*, 2024. https://www.criteo.com/blog/privacy-sandbox-testing-results-show-shortfalls-to-meet-cma-requirements/, Accessed: 2024-08-26.

[14] W.O.S. Project: *Webkit standards positions: Topics api*, 2024. https://github.com/WebKit/standards-positions/issues/111, Accessed: 2024-08-26.

[15] T. Qi, F. Wu, C. Wu, Y. Huang, and X. Xie: *Fedrec: Privacy-preserving news recommendation with federated learning.* arXiv: Information Retrieval, 2020. https://api.semanticscholar.org/CorpusID:214641339.

[16] S. Ru, B. Zhang, Y. Jie, C. Zhang, L. Wei, and C. Gu: *Graph neural networks for privacy-preserving recommendation with secure hardware.* In *2021 International Conference on Networking and Network Applications (NaNA)*, pp. 395–400, 2021.

[17] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang: *Deep graph library: A graph-centric, highly-performant package for graph neural networks*, 2020. https://arxiv.org/abs/1909.01315.

[18] C. Wu, F. Wu, L. Lyu, T. Qi, Y. Huang, and X. Xie: *A federated graph neural network framework for privacy-preserving personalization.* Nature Communications, 13(1), June 2022, ISSN 2041-1723. http://dx.doi.org/10.1038/s41467-022-30714-9.

[19] F. Wu, Y. Qiao, J.H. Chen, C. Wu, T. Qi, J. Lian, D. Liu, X. Xie, J. Gao, W. Wu, and M. Zhou: *MIND: A large-scale dataset for news recommendation.* In D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault (eds.): *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 3597–3606, Online, July 2020. Association for Computational Linguistics. https://aclanthology.org/2020.acl-main.331.

[20] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui: *Graph neural networks in recommender systems: A survey.* ACM Comput. Surv., 55(5), dec 2022, ISSN 0360-0300. https://doi.org/10.1145/3535101.

[21] J. Yi, F. Wu, C. Wu, R. Liu, G. Sun, and X. Xie: *Efficient-fedrec: Efficient federated learning framework for privacy-preserving news recommendation*, 2023. https://arxiv.org/abs/2109.05446.

[22] R. Ying, R. He, K. Chen, P. Eksombatchai, W.L. Hamilton, and J. Leskovec: *Graph convolutional neural networks for web-scale recommender systems.* In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18. ACM, July 2018. http://dx.doi.org/10.1145/3219819.3219890.

[23] G. Zhang, D. Li, H. Gu, T. Lu, and N. Gu: *Heterogeneous graph neural network with personalized and adaptive diversity for news recommendation.* ACM Trans. Web, 18(3), may 2024, ISSN 1559-1131. https://doi.org/10.1145/3649886.

A Additional results of GRAPHITEE

The following graphs show the evolution of the MRR, nDCG@5, and nDCG@10 during training GRAPHITEE-RB with random batches of size 2000. They are the completing plots of Figure 5.6.

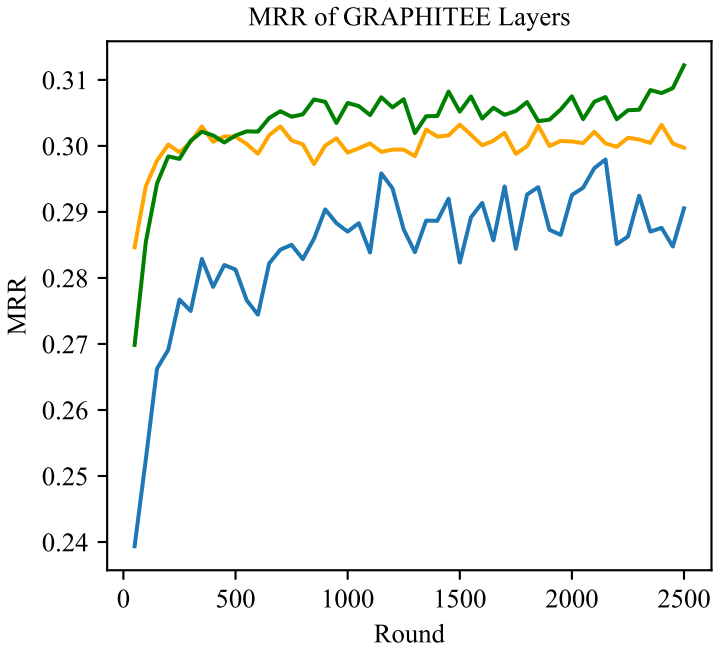

Figure A.1: Evolution of validation MRR during training. The three layers of GRAPHITEE are blue, orange, and green, in order.

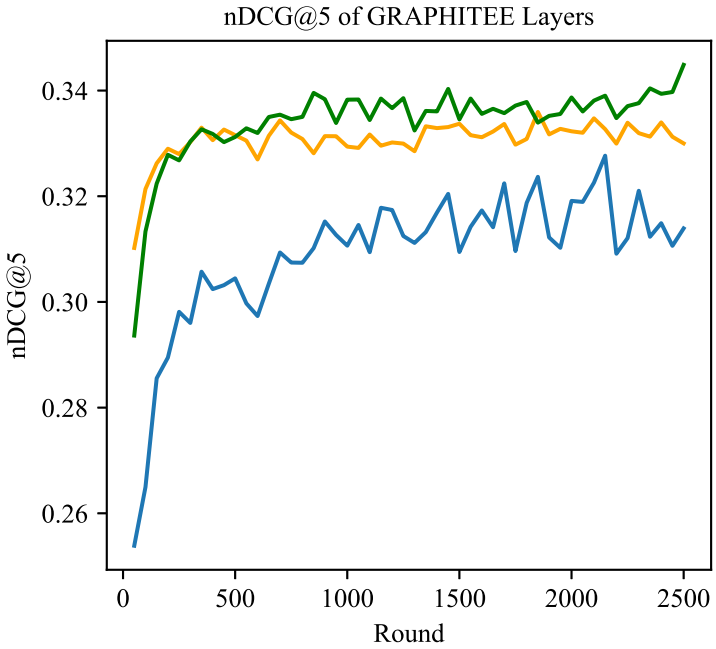

Figure A.2: Evolution of validation nDCG@5 during training. The three layers of GRAPHITEE are blue, orange, and green, in order.

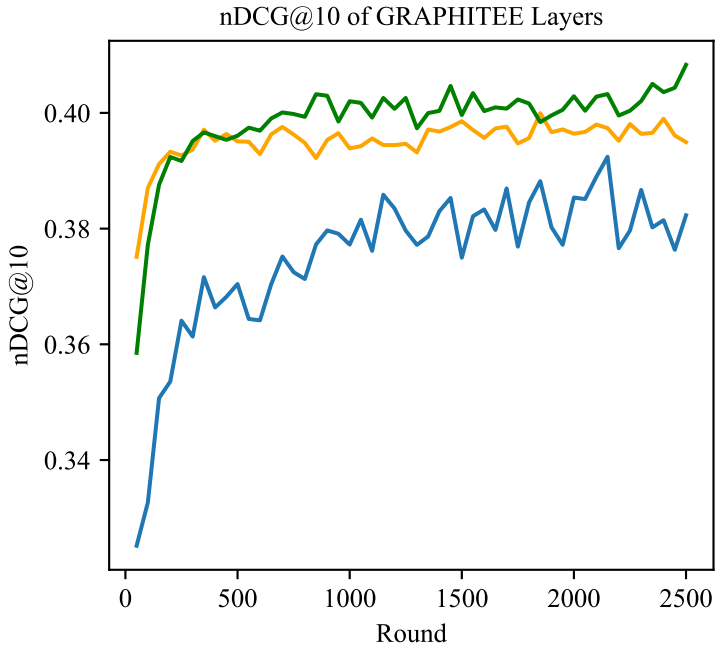

Figure A.3: Evolution of validation nDCG@10 during training. The three layers of GRAPHITEE are blue, orange, and green, in order.