FedRule: Federated Rule Recommendation System with Graph Neural Networks

YUHANG YAO*†, Electrical and Computer Engineering Carnegie Mellon University,

MOHAMMAD MAHDI KAMANI*, AI Team Wyze Labs,

ZHONGWEI CHENG, AI Team Wyze Labs,

LIN CHEN, AI Team Wyze Labs,

CARLEE JOE-WONG, Electrical and Computer Engineering Carnegie Mellon University,

TIANQIANG LIU, AI Team Wyze Labs,

Much of the value that IoT (Internet-of-Things) devices bring to "smart" homes lies in their ability to automatically trigger other devices' actions: for example, a smart camera triggering a smart lock to unlock a door. Manually setting up these rules for smart devices or applications, however, is time-consuming and inefficient. Rule recommendation systems can automatically suggest rules for users by learning which rules are popular based on those previously deployed (e.g., in others' smart homes). Conventional recommendation formulations require a central server to record the rules used in many users' homes, which compromises their privacy and leaves them vulnerable to attacks on the central server's database of rules. Moreover, these solutions typically leverage generic user-item matrix methods that do not fully exploit the structure of the rule recommendation problem. In this paper, we propose a new rule recommendation system, dubbed as **FedRule**, to address these challenges. One graph is constructed per user upon the rules s/he is using, and the rule recommendation is formulated as a link prediction task in these graphs. This formulation enables us to design a federated training algorithm that is able to keep users' data private. Extensive experiments corroborate our claims by demonstrating that **FedRule** has comparable performance as the centralized setting and outperforms conventional solutions.

Additional Key Words and Phrases: Federated Learning, Rule Recommendation, Graph Neural Networks

1 INTRODUCTION

With the rapid expansion of smart devices and applications in recent years, it becomes imperative to automate the actions of different devices and applications by connecting them together. For example, an occupancy sensor change can trigger a smart thermostat to turn on, or a code merge can trigger software updates. These connections are broadly construed as *rules* between entities in different systems. Setting up these rules between many entities can involve a tedious and challenging search process, especially for new users. Hence, it is helpful to provide new users with meaningful recommendations by learning from other users' sets of rules.

Recommendation systems are growing in various applications from e-commerce [25] to social networks [36] and entertainment industries [2, 13]. In most cases, the recommendation problem is formulated as a matrix completion

---

*Both authors contributed equally to this research.

†This work was done during Yuhang Yao's internship at Wyze Labs.

(a) User-Item Matrix Design   (b) Graph Structure Design
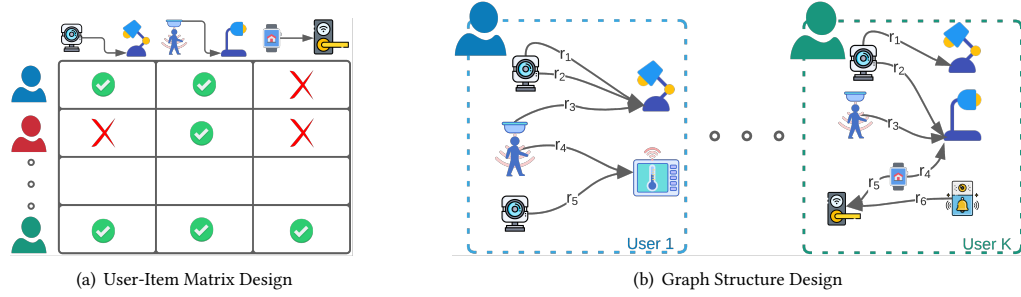
Fig. 1. Illustrations of the conventional structure of recommendation systems based on user-item structure (left), and the proposed graph-based structure for rule recommendation system in a smart home device connections setting (right). In the conventional setting, each rule is considered an item while in the graph structure each rule is represented as an edge between pairs of entity nodes.

task of a user-item matrix [3, 9, 38] to recommend new items to the users based on previous items chosen by the user. In this way, the recommendation problem is reduced to a dual link between users and items, as depicted in Figure 1(a). Following this setting, we consider each rule as an item for recommending linked rules to users. However, this is not desirable as the structure of entities (*e.g.* the structure of devices or applications owned by a user) on the user side is not considered. Hence, we are not able to distinguish between different instances of an entity type (*e.g.* multiple cameras or multiple motion sensors) to provide meaningful recommendations for each separate entity based on their existing structure. For instance, in a smart home network, there could be multiple contact sensors connected to a different set of devices in their vicinity. Without considering the existing structure of the connected devices, the recommendation system tends to recommend the same rules for all contact sensors as they are recognized as one general entity instead of several independent ones. Moreover, *privacy* and *security* concerns for training models on users' data in recommendation systems formulated as user-item matrix completion become a great challenge in the IoT system. The user-item formulations require a central server to know all users' rules: these may be sensitive, especially if they involve smart devices' behavior in users' homes.

In this paper, we, first, propose a new rule recommendation framework dubbed as `GraphRule`, based on the graph structure of entities for each user. In our proposed setup, we create a directed graph for each user based on their available entities (*e.g.* light bulb and contact sensor) as nodes and their rule connections as edges in this graph. Therefore, instead of representing a user's rules as a row in the user-item matrix (each item is an entity-rule-entity triplet, *e.g.* when contact sensor is open, turn on light bulb), we represent each users' rules as a graph that encapsulates the structure of entities and how they are connected together through the available rules. As an example, in Figure 1(b), for user 1, we have two separate cameras denoted by different nodes in the graph, each connecting to a distinct set of devices by their specific rules. However, it is infeasible to distinguish between these cameras in the user-item structure without separating the rule and entities representations from the entity-rule-entity items in user-item matrices. In fact, in user-item design, we condense entities with the same type into one general entity (*e.g.* camera 1 and camera 2 into the camera) to only emphasize their connections to other entities as rules. Based on our graph structure design, the goal of the rule recommendation system is then to predict the newly formed edges (rules) in the graph, which can be formulated as a graph link prediction problem [43]. The system first learns node embeddings for each entity in the user's graph using graph neural network architectures [31, 35, 39]. Based on the learned embeddings, a parametric prediction model can be used to estimate the probability of different rules connecting each pair of entities in the graph.

For privacy concerns, unlike the conventional user-item structure, the proposed graph structure can be easily distributed across different users for training the main recommendation model locally. Indeed, by casting the rule recommendation problem as a link prediction task across graphs for different users, we can employ privacy-preserving training setups such as federated learning [27], which allows data to remain at the user's side. Moreover, federated training methods are generally iterative, and can thus easily adapt the model to new users' data instead of having to retrain the whole model from scratch, as is common in previous approaches for recommendation systems.

There has been a surge in applications of federated learning in the training of machine learning models recently [5, 10, 11, 14, 19, 27], with some applications in recommendation systems [8, 37]. However, most of the proposals for recommendation tasks in federated learning are for *cross-silo* setups (*i.e.*, data could be distributed across pre-defined silos) based on user-item matrix completion, and not for *cross-device* settings [19] (*i.e.*, with limited data at each user, where here "device" represents one user and its entities), as in our rule recommendation task.

Our cross-device setting with the proposed graph structure, however, introduces new challenges: the amount of data on each individual user is very limited (as each user generally has only a few entities, e.g., a few smart home devices) and the local data is not independent and identically distributed (non-IID). The non-IID data distribution with a small sample size severely increases the variance of gradients among clients. Hence, when simply applying federated training methods, like FedAvg [16, 18, 27], the model will not converge easily and cannot have comparable performance on par with the centralized training. To overcome the issue of severely non-IID data distribution across clients, we use two control parameters in local machines for each client to correct the gradients for different parts of our model and avoid drifting too much from the average model. Given the limited data of users, we sample negative edges (rules that do not exist) to balance the numbers of positive and negative edges. With these solutions, we then introduce **FedRule**, a federated rule recommendation system with graph neural networks. **FedRule** can learn the representation and link prediction models of the recommendation system at the user level with a decent convergence rate using variance reduction techniques with control parameters while preserving the privacy of users' data. With negative sampling and variance reduction to alleviate the effects of limited and non-IID data, this system enables us to learn a better model in a graph-based cross-device setting over user-item-based settings. In our empirical results, we show the effectiveness of our **FedRule** over other approaches.

Our main contributions are:

- We propose a new rule recommendation framework, called **GraphRule**, based on the graph structure of current users by representing rules as edges between entity nodes over conventional user-item structure.
- We propose an effective federated learning system, called **FedRule**, which effectively learns from limited non-IID data on each user's rules with fast convergence and preserving the privacy of users' data.
- Extensive empirical investigation shows the effectiveness of both the proposed structure for the **GraphRule** and the **FedRule** algorithms. Our **FedRule** algorithm is able to achieve the same performance as the centralized version (**GraphRule**) due to its variance reduction mechanism.

2 RELATED WORK

Rule Recommendation Systems. Traditional recommendation systems are mainly based on matrix factorization techniques [4, 22] and aim to match users to a set of items they are interested in, based on the features of users and items as well as prior data expressing users' interests in a given set of items. Recently, graph neural networks have been widely adopted by many recommendation systems to model this relationship as a bipartite graph structure. GCMC [3] provides a graph-based auto-encoder framework for matrix completion problems. Given different relations between

users and items, R-GCNs [32] models relational data with graph convolutional networks and HetGNN [42] is to deal with heterogeneous graphs with features of users and items. RotatE [33] uses knowledge graph embedding by relational rotation in complex number space.

With the development of the smart home technology [41], rule recommendation systems for IoT devices exhibit great practical and research potential. IFTTT, a mobile app providing rule execution services for mobile applications and IoT devices, provides open access data with users and rules for analysis [28, 34]. For recommending rules in IFTTT, RecRule [9] provides rule recommendation based on a user-rule matrix with a semantic reasoning process to enrich rules with semantic information of rules, and TaKG [38] builds a trigger-action knowledge graph with the user-rule collaborative filtering to recommend rules.

Graph Neural Networks. Graph neural networks aim to learn representations of graphically structured data that capture features associated with graph nodes as well as edges between the nodes [6]. Recently graph neural networks, Graph Convolutional Networks (GCN) [21], GraphSage [17], and Graph Attention Networks (GAT) [35], are proposed and show great performance on various graph learning tasks. A popular task is that of link prediction [43], i.e., predicting whether a link exists between two nodes in the graph, given information on the presence or absence of links between other pairs of nodes. We show that the rule recommendation problem can be formulated as a link prediction one, and go beyond prior work on GNNs for link prediction by considering a federated solution algorithm that can be learned in a distributed way across multiple users.

Federated Learning. Federated learning is first proposed in [27], which allows clients to train a common machine learning model on their collective data while keeping personal data on clients. Instead of uploading data to the server for centralized training, clients iteratively process their local data and share model updates with the server. Weights from a large population of clients are periodically aggregated by the server and combined to create an improved global model. The widely adopted FedAvg algorithm [27] is used on the server to combine client updates and produce a new global model. However, FedAvg may fail to converge if data from different clients follow different distributions (i.e., is non-IID), as is often the case in federated learning scenarios, including our rule recommendation system. To deal with the non-IID local data distribution [24, 45], various solutions such as SCAFFOLD [20], FedProx [23], and more recently, FedGate [15] are proposed to reduce the variance of data distribution and speed up the learning.

Recent work proposes federated learning algorithms specialized to graph neural networks, and the model aims to predict characteristics of the graph, such as link prediction, node classification, and graph classification. FedGraphNN [18] provides a federated learning system and benchmark for graph neural networks, which mainly focuses on system deployment. FedGCN [40] focuses on reducing the communication cost of training a large graph with cross-client edges. Both work only uses FedAvg for the training process. Our work, however, focuses on learning techniques to handle the challenges of limited and non-IID data in rule recommendation problems.

There are also algorithms that aim to use federated learning to make recommendations. FedMF [7] provides secure federated matrix factorization. FCF [1] provides federated collaborative filtering for privacy-preserving personalized recommendation systems. Wu et al. [37] uses a federated heterogeneous graph neural network to model user-item relationships. All these federated recommendation systems use the user-item matrix formulation, which requires communication among clients to exchange user information and is not fully federated, potentially raising privacy and security concerns. By re-formulating the rule recommendation problem to a link prediction task in a graph, we eliminate the need for communication among clients, helping to preserve user privacy and reducing the overhead of communication during training.

3 FEDERATED RULE RECOMMENDATION

Recommendation systems often rely on gathering data from all users and their interactions with items or other users to learn prediction models. This procedure poses a great threat to user privacy, as seen in the Netflix Recommendation challenge, where researchers were able to identify individuals by matching Netflix profiles with those of IMDB profiles [29]. Hence, it is of paramount importance to consider privacy-preserving systems for recommendation services. Federated Learning [27] provides a viable solution for training such models without the need to move users' data to a server. Thus, it can be employed in training these models on user-item or user-user interaction data.

The main challenge that hinders the applications of federated learning in recommendation systems is the structure of data in such systems, which cannot be distributed easily. For instance, matrix factorization or even recent algorithms such as Graph Convolutional Matrix Completion (GCMC) [3] require access to the full user-item matrix they intend to complete. Although some distributed variations of these algorithms have been proposed [12, 26], they are mostly focused on computational efficiency in a structured homogeneous distribution of data. Thus, it addresses neither user privacy concerns nor heterogeneity of data across the users' network. For the rule recommendation problem, we address these concerns by proposing a federated learning setup based on the graph structure of rules. Despite our focus on rule recommendation with graph link prediction, the proposed learning approach can evidently be applied to other tasks such as graph classification [44] and node classification [21].

3.1 Graph Rule Recommendation

In common recommendation systems, the goal is to predict unknown links between users and items. However, in rule recommendation problems we want to suggest how users can connect two or more entities (*e.g.* smart devices) together. In previous approaches, we are limited to considering multiple entities' connections as the connections of one abstract entity to create a user-item matrix (with items as rules). In this case, the item is a rule between those entities, and hence, we are losing information on the graph structure of entities and their connections.

In the rule recommendation problem, similar to the graph link prediction problem [43], we need to consider the relationship between different entities as well. Hence, instead of representing the connections as a row in a 2D matrix of user-item relationships, we have a graph of entities' connections for each user (see Figure 1). The graph also enables us to represent entities with the same type (*i.e.*, Cameras) as separate entity nodes in the graph. In addition, in this setting, compared to a simple graph, instead of binary connections between different entities, we can have multiple types of edge connections. This will enable us to represent different types of connections as different rules between entities. Using this structure, we can leverage advancements in the graph neural network domain to learn a better model for recommending new rules to the users.

Therefore, given the set of users $\mathcal{U}$ and the set of connection types $\mathcal{R}$, for each user $u_k \in \mathcal{U}$ in our setting, we create a graph of entities denoted by $\mathcal{G}_k = (\mathcal{V}_k, \mathcal{X}_k, \mathcal{E}_k)$, where $\mathcal{V}_k$ is a set of nodes (entities), $\mathcal{X}_k$ is the set of nodes' (entities') features (*e.g.*, types of entities), and $\mathcal{E}_k$ is the set of connections between nodes in this graph. $\mathcal{X}_k$ can be represented as $\{\mathbf{x}_v, \forall v \in \mathcal{V}_k\}$, where $\mathbf{x}_v$ represents the feature vector of node v. Each connection in $\mathcal{E}_k$ can be represented as (v_i, v_j, r), which means there is an edge (rule) between node v_i and node v_j with connection type $r \in \mathcal{R}$ (e.g. "is open, turn on" in "when the contact sensor is open turn on the light"). The goal of the recommendation system in this problem is to learn an embedding model $\boldsymbol{\theta}$ for nodes in the graph of user u_k and a predictor $\boldsymbol{\phi}$ that can use the node embeddings to estimate the following probability:

$$\mathbb{P}\big((v_i, v_j, r) \in \mathcal{E}_k | \boldsymbol{\theta}, \boldsymbol{\phi}; \mathcal{G}_k\big). \tag{1}$$

Algorithm 1: GraphSage Node Embedding Generation Algorithm for Client u_k

---

Inputs: Graph $\mathcal{G}_k = (\mathcal{V}_k, \mathcal{X}_k, \mathcal{E}_k)$, where input features $\mathcal{X}_k = \{\mathbf{x}_v, \forall v \in \mathcal{V}_k\}$; Weight matrices θ^1 and θ^2;
  Differentiable aggregator function AGG (.); Neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}_k}$.
Output: Node embeddings $\mathbf{z}_v$ for all $v \in \mathcal{V}_k$.
// Node Features as Hidden State
$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}_k$;
// First GraphSage Layer with ReLU Activation
for $v \in \mathcal{V}_k$ **do**
  $\mathbf{h}_{\mathcal{N}(v)}^1 \leftarrow \text{AGG}(\{\mathbf{h}_{v_n}^0, \forall v_n \in \mathcal{N}(v)\})$;
  $\mathbf{h}_v^1 \leftarrow \text{ReLU}(\theta^1 \cdot \text{CONCAT}(\mathbf{h}_v^0, \mathbf{h}_{\mathcal{N}(v)}^0))$;
end
// Second GraphSage Layer
for $v \in \mathcal{V}_k$ **do**
  $\mathbf{h}_{\mathcal{N}(v)}^2 \leftarrow \text{AGG}(\{\mathbf{h}_{v_n}^1, \forall v_n \in \mathcal{N}(v)\})$;
  $\mathbf{h}_v^2 \leftarrow \theta^2 \cdot \text{CONCAT}(\mathbf{h}_v^1, \mathbf{h}_{\mathcal{N}(v)}^1)$;
end
$\mathbf{z}_v^k \leftarrow \mathbf{h}_v^2, \forall v \in \mathcal{V}_k$

Based on this probability we can then recommend new edges to the users.

3.2 Model

For each client, our rule recommendation model has two parts: a Graph Neural Network (GNN) to calculate the node embeddings and a predictor to predict the connections between nodes (the types of rules) as well as their probabilities. As discussed above, each client u_k has a graph $\mathcal{G}_k$ to represent the ground truth connections between different nodes.

3.2.1 GNN. We choose a two-layer graph neural network to get the embedding of nodes. Algorithm 1 shows the GraphSage [17] model that we used. In each GraphSage layer l, at first the differentiable aggregator function $\text{AGG}(\{\mathbf{h}_{v_n}^l, \forall v_n \in \mathcal{N}(v)\})$ aggregates information from node neighbors, where $\mathbf{h}_{v_n}^l$ denotes the hidden state of node v_n and $\mathcal{N}(v)$ denotes the set of neighbors of node v. We use the mean operator as the aggregator function which takes the element-wise mean of the hidden states. Then, weight matrices θ^1 and θ^2 are used to propagate information between different layers of the model. After the two-layer GNN, we can then get the node embeddings $\mathbf{z}_v^k$ for all $v \in \mathcal{V}_k$.

3.2.2 Predictor. For each client u_k, after we get the node embeddings $\mathcal{Z}_k = \{\mathbf{z}_v^k, \forall v \in \mathcal{V}_k\}$. We then need to predict the edge connection probability $\mathbf{p}_{v_i, v_j}^k \in [0, 1]^{|\mathcal{R}|}$ between node v_i and node v_j for all $v_i \in \mathcal{V}_k$ and all $v_j \in \mathcal{V}_k$. In our case, for the predictor model, we use a two-layer fully-connected neural network with ReLU activation for the first layer and the Sigmoid function in the last layer. The predictor uses $\mathbf{z}_{v_i}^k$ and $\mathbf{z}_{v_j}^k$, embeddings of node v_i and node v_j as input, makes it pass the 2-layer neural network (ϕ) with weight matrices ϕ^1 and ϕ^2, then outputs the link probability of edges.

3.2.3 Centralized Optimization. The model can be easily trained in a centralized setting where the centralized server stores all user graphs $\mathcal{G}_k = (\mathcal{V}_k, \mathcal{X}_k, \mathcal{E}_k)$ for all $u_k \in \mathcal{U}$. We call this centralized training on the graph structure for rule recommendation as **GraphRule**. User graphs are mainly sparse, meaning most of the connections between nodes are not set yet. Given the number of possible types of edges between nodes, there are only a few positive edges $\mathcal{E}_k^{\text{pos}}$ where the positive edge means that there is a formed connection between the two entities. The other possible edges can

then be considered as negative edges $\mathcal{E}_k^{\text{neg}}$. Considering all negative edges lengthens the training time, so we *sample negative edges* to balance the numbers of positive and negative edges.

We then use binary cross entropy loss including positive and negative edges as our objective function:

$$\mathcal{L}_k(\mathbb{P}^k, \mathcal{E}_k^{\text{pos}}, \mathcal{E}_k^{\text{neg}}) = -\frac{1}{|\mathcal{E}_k^{\text{pos}}| + |\mathcal{E}_k^{\text{neg}}|} \left(\sum_{e \in \mathcal{E}_k^{\text{pos}}} \log(p_e) + \sum_{e \in \mathcal{E}_k^{\text{neg}}} \log(1 - p_e) \right), \tag{2}$$

where $\mathbb{P}^k \in [0,1]^{|\mathcal{V}_k| \times |\mathcal{V}_k| \times |\mathcal{R}|}$ denotes all pairs of edge connection probability in $\mathcal{G}_k$, and $p_e \in [0,1]$ denotes the connection probability of a specific edge $e = (v_i, v_j, r)$.

By denoting the GNN as $\boldsymbol{\theta}$ and predictor as $\boldsymbol{\phi}$, the loss given user graph $\mathcal{G}_k$ can then be represented as

$$\mathcal{L}_k(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathcal{G}_k) = \mathcal{L}_k(\mathbb{P}^k(\boldsymbol{\theta}, \boldsymbol{\phi}), \mathcal{E}_k^{\text{pos}}, \mathcal{E}_k^{\text{neg}}). \tag{3}$$

At each iteration, the centralized server performs gradient descent by calculating the loss

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathcal{G}_1, .., \mathcal{G}_{|\mathcal{U}|}) = \sum_{u_k \in \mathcal{U}} \mathcal{L}_k(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathcal{G}_k), \tag{4}$$

then updates the model.

3.3 FedRule

Learning the aforementioned models at scale requires gathering the graph data of every user at a server to run the training. The `GraphRule` training, despite its fast convergence speed, can expose users' private data related to the devices or applications they are using and how they are connected together. Hence, it is important to facilitate a privacy-preserving training procedure to safeguard users' data. Federated learning [27] is the de-facto solution for such purpose in distributed training environments.

Recently, there have been some proposals to apply federated learning in recommendation problems such as in [8, 30, 37]. However, almost all these proposals are for cross-silo federated learning and are not as granular as cross-device federated learning. In this paper, the problem of rule recommendation is formulated so as to be more suited for the cross-device federated learning setup. Although the cross-device federated learning setup is more desirable for the purpose of privacy-preserving algorithms, it makes the training procedure more challenging. The reason behind this is that for the problem of rule recommendation, the size of the data (*i.e.*, the graph structure for each client) is small, and it follows non-IID distribution due to heterogeneous user behaviors. The non-IID data distribution increases the variance of gradients among users and makes the gradient updates coming from different users go in different directions. Hence, the local training in federated learning by averaging the gradients is hard to converge due to misaligned directions of gradients. As it is shown in Section 4, applying FedAvg with GNN on cross-device settings [18, 27] like rule recommendation problem can fail in some cases due to the non-IID problem mentioned above.

We then propose the `FedRule`, federated rule recommendation system with graph neural network. The design schema of the system is depicted in Figure 2. We use negative sampling, which samples negative edges in the graph, to balance the numbers of positive and negative edges. To avoid the non-IID problem, we use two control parameters in local machines for each client to correct the gradients and avoid drifting too much from the average model. `FedRule` as presented in Algorithm 2 consists of four main parts as follows:
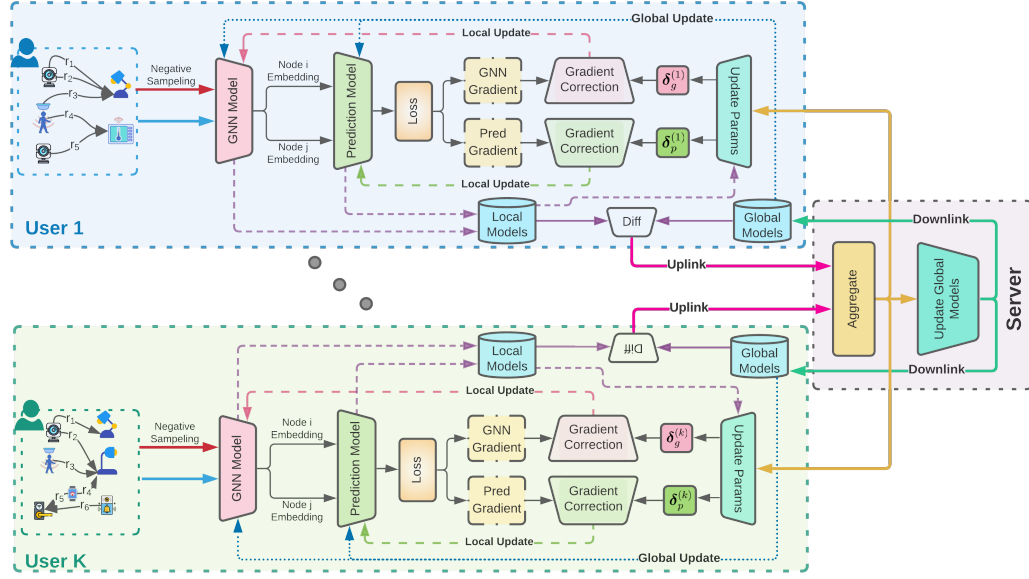
Fig. 2. Overview of **FedRule** System Architecture Design.

Local Updates. At the beginning of each local training stage (communication round c), clients will get the updated global GNN (θ_k) and prediction (ϕ_k) models. Then, in each local iteration t, the client's device computes the gradients of models using local data. The gradient for the GNN part is with respect to the graph data ($\mathcal{G}_k$) and for the prediction model is with respect to the set of node embeddings ($\mathcal{Z}_k$) from the local graph. With the control parameters described next, the gradients get corrected and then the local models ($\theta_k^{(t,c)}, \phi_k^{(t,c)}$) get updated using their respective learning rates (η_θ, η_ϕ). Note that since the GNN model is a global representation of nodes and the prediction model is more of a personalized classifier, the learning rates of the models might be different.

Gradient Correction. For each local iteration, after computing the gradients, we adapt the FedGATE algorithm [15] and use two control parameters ($\delta_{\theta_k}, \delta_{\phi_k}$) for GNN model and predictor model respectively. The use of these control parameters will help the training to reduce the variance of convergence, as it can be inferred from the experimental results in Section 4 as well. Similar to learning rates, due to the different natures of the models, they might get corrected with different rates using parameters $\lambda_\theta, \lambda_\phi$.

Model Aggregation. After τ_k local steps in each client's device, we aggregate the models from the devices. To do so, we first compute the difference between the current local models and the starting global models at round c, denoted by $\Delta_{\theta_k}^{(\tau_k,c)}, \Delta_{\phi_k}^{(\tau_k,c)}$. Then, the server averages over these updates from clients and send back these averages to the clients ($\Delta_\theta^{(c)}, \Delta_\phi^{(c)}$). Also, the server uses these average updates to update the global models that need to be broadcast to the clients in the next round. Secure gradient aggregation methods can be integrated into the system to better protect privacy.

Algorithm 2: FedRule Federated Learning for Rule Recommendation Systems

---

for $c = 1, \ldots, C$ **do**

    for *each client* $k \in [K]$ **do in parallel**

        Set $\theta_k^{(1,c)} = \theta^{(c)}, \phi_k^{(1,c)} = \phi^{(c)}$,

        for $t = 1, \ldots, \tau_k$ **do**

            Set $g_{\theta_k}^{(t,c)} = \nabla_{\theta_k} \mathcal{L}_k(\theta_k^{(t,c)}, \phi_k^{(t,c)}; \mathcal{G}_k)$

            Set $g_{\phi_k}^{(t,c)} = \nabla_{\phi_k} \mathcal{L}_k(\theta_k^{(t,c)}, \phi_k^{(t,c)}; \mathcal{Z}_k)$

            // Correct Gradients

            $\tilde{g}_{\{\theta_k, \phi_k\}}^{(t,c)} = g_{\{\theta_k, \phi_k\}}^{(t,c)} - \lambda_{\{\theta, \phi\}} \delta_{\{\theta_k, \phi_k\}}$

            // Update Parameters

            $\theta_k^{(t+1,c)} = \theta_k^{(t,c)} - \eta_\theta \, \tilde{g}_{\theta_k}^{(t,c)}$

            $\phi_k^{(t+1,c)} = \phi_k^{(t,c)} - \eta_\phi \, \tilde{g}_{\phi_k}^{(t,c)}$

        end

        // Update Control Parameters

        send $\Delta_{\{\theta_k, \phi_k\}}^{(\tau_k, c)} = \left\{ \theta_k^{(c)}, \phi_k^{(c)} \right\} - \left\{ \theta_k^{(\tau_k, c)}, \phi_k^{(\tau_k, c)} \right\}$ to server and gets $\Delta_{\{\theta, \phi\}}^{(c)}$

        Update control parameters using Eq. (5)

    end

    // Server Operations

    // Difference Aggregation

    $\Delta_{\{\theta, \phi\}}^{(c)} = \frac{1}{K} \sum_{k=1}^{K} \Delta_{\{\theta_k, \phi_k\}}^{(\tau_k, c)}$ and broadcasts back to clients

    // Update Global Models

    Compute $\left\{ \theta^{(c+1)}, \phi^{(c+1)} \right\} = \left\{ \theta^{(c)}, \phi^{(c)} \right\} - \Delta_{\{\theta, \phi\}}^{(c)}$ and broadcast to local clients

end

Parameter Updates. Using the calculated average updates in the previous stage, clients update their local control parameters using the deviation of the local updates from average updates:

$$\delta_{\theta_k}^{(c+1)} = \delta_{\theta_k}^{(c)} + \frac{1}{\eta_\theta \tau_k} \left(\Delta_{\theta_k}^{(\tau_k, c)} - \Delta_{\theta}^{(c)} \right)$$

$$\delta_{\phi_k}^{(c+1)} = \delta_{\phi_k}^{(c)} + \frac{1}{\eta_\phi \tau_k} \left(\Delta_{\phi_k}^{(\tau_k, c)} - \Delta_{\phi}^{(c)} \right) \tag{5}$$

4 EMPIRICAL EVALUATION

After introducing our recommendation formulation, model, and training algorithm, we empirically evaluate the proposed algorithm. In this section, we first introduce the two datasets used in our experiments and comparison methods, then compare our methods in both the centralized and federated settings with baseline algorithms. After that, we show that our proposal can handle multiple entities of the same type cases efficiently.

4.1 Datasets

Wyze Smart Home Rules Dataset. To experiment the efficacy of our proposed algorithm, we first use a real-world dataset for smart home devices from the Wyze Labs'[1] rule engine. This dataset contains the rules that connect smart devices in different clients' houses. Hence, by nature, the distribution of rules among different clients is non-IID, which

---

[1]https://www.wyze.com/

is in line with the federated learning setting. We call this dataset the "*Wyze Smart Home Rules*" dataset, which contains $76,218$ users with $201,940$ rules. We cleaned the dataset to remove all personally identifiable information (PII) in order to address privacy concerns. We simplify the current rules into the following form: < trigger entity, trigger-action pairs, action entity>, where the trigger-action pairs denote the connection type. For instance, we can connect a smart doorbell to another camera, when by pressing the doorbell we want to power on the camera for recording. Then, the rule format is < Doorbell, Doorbell Pressed - Power On, Camera>. We have 11 unique entities and 163 unique trigger-action pairs, resulting in a total of $1,207$ unique rules. The types of entities and trigger-action pairs are shown in Table 1.

IFTTT Dataset. The IFTTT Dataset [9] is one of the most popular EUD tools. To the best of our knowledge, this is the only publicly available dataset of IF-THEN rules defined and shared by different users. But it does not support entities of the same type. It was obtained by Ur et al. [34] with a web scrape of the IFTTT platform as of September 2016. The dataset contains 144 different users with $8,729$ rules. There are $3,020$ types of rules used by users. In the current dataset, we use 53 unique entities and 132 unique trigger-action pairs. The types of entities and trigger-action pairs are shown in Table 2.

Types of Entities	Types of Trigger-Action Pairs
Camera	Open, Power On
Chime Sensor	Open, Power Off
Contact Sensor	Open, Motion Alarm On
Light	Open, Change Brightness
Lock	...
Mesh Light	Open, Siren On
Motion Sensor	Open, Alarm Action
Outdoor Plug	...
Plug	Person Detected, Power Off
Thermostat	Smoke Detected, Power Off
Outdoor Camera	Doorbell Pressed, Power On

Table 1. Smart Home Rules Dataset entities and trigger-action pairs.

Types of Entities	Types of Trigger-Action Pairs
Android Device	New Post, Share a Link
Weather	New Follower, Post a Tweet
Gmail	New Like, Add File from Url
YouTube	New Liked Video, Create a Post
...	...
Facebook	New Photo, Send Me an Email
Instagram	New Photo, Add File from Url
Linkedin	New Screenshot, Add File from Url
...	...
Twitter	You Exit an Area, Set Temperature
Reddit	Battery Low, Send an Sms

Table 2. IFTTT Dataset entities and trigger-action pairs.

4.2 Experimental Setups

Comparing Methods. We compare these methods

- Matrix Factorization [4]: Complete user-item matrix by matrix factorization.
- GCMC [3]: Graph-based auto-encoder framework for matrix completion based on a user-item bipartite graph.
- `GraphRule`: The proposed centralized optimization for graph formulation of rule recommendation.
- FedAvg with GNN [18, 27]: Each user has a user graph and a local model. The server aggregates the local models and uses FedAvg to train the global model. We call this FedGNN.
- `FedRule`: Our proposed federated rule recommendation algorithm.

Experiment Setting. We use the Adam optimizer with a learning rate of 0.1 and 100 training rounds. For FL algorithms, we use 3 local steps at each communication round, for 300 total iterations. The dimensions of the hidden states between the two GraphSage layers and the two NN layers are 16 and the number of possible trigger-action pairs, respectively. For `GraphRule`, users' graphs are stored in the central server and we do the gradient descent with all users' graphs. For

FL methods, we compute the batch gradient descent on each user's graph with 3 local epochs to train the local models, then the server aggregates the local models to update the global model. We set the hyper-parameter $\lambda = 1$ for the FedRule algorithm. For each user, we use 80% rules set by the user for training and the remaining 20% rules for testing.

Evaluation Metrics. We use the following metrics to compare algorithms:
- Loss: Binary cross entropy loss with the positive and negative edges.
- AUC: Area Under the Curve.
- Mean Rank: Mean rank of positive testing edges between specific entities.
- Hit Rate@N: Recommend N rules and check if the positive test edges are included.

4.3 Comparison of Graph-Based and User-Item-Based Methods

Quantitative Evaluation. We first compare our centralized graph model, `GraphRule`, with user-item-based methods. For a fair comparison with user-item-based methods, we consider entities with the same type as one entity. As shown in Figure 3, the hit rate of `GraphRule` is outperforming both GCMC and Matrix Factorization in this task, which validates our analysis in Section 3.1. With the increase of the number of recommendations, `GraphRule` has a better hit rate, close to 0.91, when recommends 40 rules, which means by recommending 40 rules among 1207 rules total, the user has 91% chance to choose rules in the recommendation lists on average.

Qualitative Evaluation. To qualitatively evaluate the performance of these algorithms we provide a case, where we feed a user's rule data to each of the models while omitting one of the rules set by that
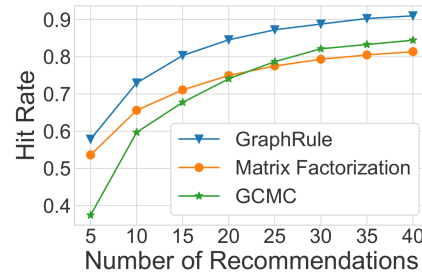

Fig. 3. Hit Rate on the test set of Smart Home Rules Dataset vs. number of recommendations, comparing `GraphRule` with conventional recommendation systems. Recommendations are filtered by valid rules for fair comparison.

user. The user's rule examples are described in Table 3. Then, we show what is the rank of that omitted rule in the recommendations made by each algorithm as well as other top recommendations in Table 4. As it can be seen in Table 4, the removed rule is the third rank in the recommendation by `GraphRule`, while it is the 17th by GCMC. It was not in any of the top 20 recommendations of the matrix factorization approach. Also, note that most of the recommendations by matrix factorization are for cameras, which is due to the popularity of camera rules among other users. Also, GCMC top rules are for a motion sensor that is not available in the user's entity set. On the other hand, all recommendations by `GraphRule` are applicable to the current set of entities for the user.

Triggers		Actions	
Entity	Trigger	Action	Entity
Contact Sensor	Open	Power On	Camera
Contact Sensor	Open	Motion Alarm	Camera
Contact Sensor	Open	Power On	Plug
Contact Sensor	Open	Power On	Light
Contact Sensor	Open	Notifications	Camera

Table 3. Arbitrary user's rules. The last rule is omitted from the users' rules for prediction.

	GraphRule		Matrix Factorization		GCMC	
Rank	Recommendation Rule	Prob.	Recommendation Rule	Prob.	Recommendation Rule	Prob.
1	(Contact Sensor, Open, Change Brightness, Light)	0.997	(Camera, Motion Detected, Alarm, Camera)	0.999	(Motion Sensor, Motion Detected, Power On, Light)	0.910
2	(Contact Sensor, Open, Alarm, Camera)	0.996	(Camera, Motion Detected, Motion Alarm, Camera)	0.980	(Motion Sensor, Motion Detected, Change Brightness, Light)	0.855
3	(Contact Sensor, Open, Notifications, Camera)	0.995	(Camera, Motion Detected, Notification, Camera)	0.972	(Motion Sensor, Being Off, Power Off, Light)	0.840
	⋮		⋮		⋮	
17	(Contact Sensor, Being Open, Notification On, Camera)	0.882	(Camera, Sound Detected, Notification, Cloud)	0.335	(Contact Sensor, Open, Notifications, Camera)	0.623
18	(Contact Sensor, Condition Met, Alarm, Camera)	0.860	(Camera, Condition Met, Power On, Camera)	0.322	(Motion Sensor, Condition Met, Power On, Light)	0.592
19	(Contact Sensor,Being Open , Power On, Camera)	0.860	(Camera, Condition Met, Motion Alarm, Camera)	0.321	(Camera, Motion Detected, Alarm, Camera)	0.552
20	(Contact Sensor, Being Open, Alarm, Camera)	0.851	(Camera, Condition Met, Notification, Camera)	0.315	(Camera, Person Detected, Notification, Cloud)	0.541

Table 4. Qualitative comparison of **GraphRule**, Matrix Factorization, and GCMC methods in predicting the removed rule from the rule set of an arbitrary user in Table 3. The removed rule is the third recommendation in **GraphRule**, but it is the 17th one in GCMC. The removed rule is not included in the first 20 rules recommended by matrix factorization.
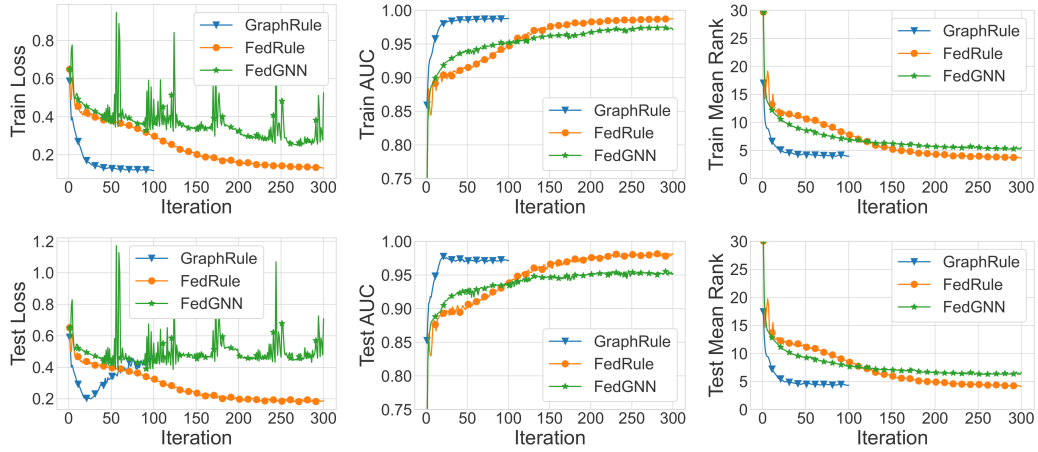

Fig. 4. Train and test performance of different algorithms on Wyze Smart Home Rules Dataset. **FedRule** smoothly converges while FedGNN training diverges after a while.

4.4 Evaluation of Centralized and Federated Algorithms

In this part, we evaluate the performance of our proposed **FedRule**, and compare it with FedGNN training, as well as our proposed centralized **GraphRule**. We perform the evaluations on both train and test datasets.

Results on Wyze Smart Home Rules Dataset. The results of applying the aforementioned algorithms on the Wyze Smart Home Rules dataset are depicted in Figure 4. During the training, the centralized algorithm, **GraphRule**, converges faster since it utilizes the training data of all users in an IID manner. For federated settings, given the heterogeneous data distribution among users and the heterogeneity of the number of rules and entities for each user, FedGNN diverges after 50 iterations. This is especially exacerbated by a large number of users with a small amount of data (76, 218 users and 2.65 rules per user on average). On the other hand, **FedRule** smoothly converges to the same loss as the **GraphRule** due to its variance reduction mechanism. The test loss shows that even **GraphRule** overfits after 20 iterations and FedGNN diverges after 50 iterations. The test loss of **FedRule** smoothly decreases and converges to the same value as the minimum loss in **GraphRule** after 200 iteration. Similarly, the test AUC of **GraphRule** converges faster. During the training process, FedGNN has a better test AUC at the start but increases slowly after 50 iterations as the training loss
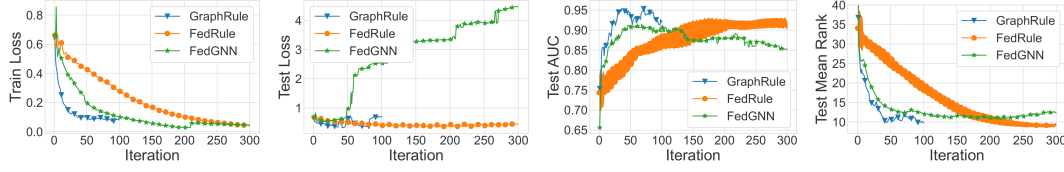
Fig. 5. Training and test performance of different centralized and federated algorithms on IFTTT Dataset. Again, **FedRule** smoothly converges while FedGNN overfits.
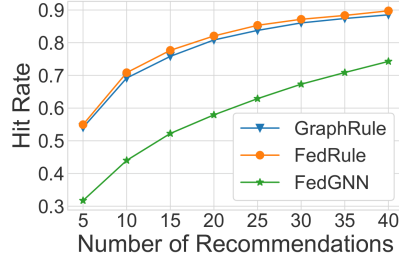

Fig. 6. Hit Rate on the test set of Smart Home Rules Dataset vs. number of recommendations, comparing **GraphRule** with federated approaches.
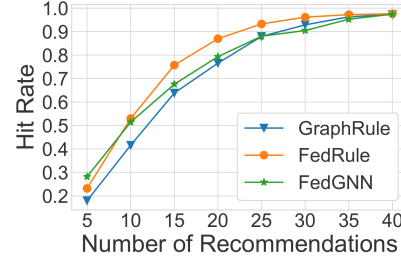
Fig. 7. The Hit Rate on the IFTTT test dataset for centralized and federated algorithms with the rule filter.

	Loss	AUC	MR	MR(RT)
GraphRule	0.1997	0.9768	4.349	3.0970
FedGNN	0.3878	0.9521	6.661	5.3946
FedRule	**0.1892**	**0.9804**	**4.156**	**2.9150**

Table 5. Final test results on Smart Home Rules Dataset (RT: remove rules shown in training graph).

	Loss	AUC	MR	MR(RT)
GraphRule	**0.3238**	**0.9491**	9.6997	9.6745
FedGNN	0.4905	0.9072	12.4479	12.4240
FedRule	0.3614	0.9417	**8.8812**	**8.8564**

Table 6. Final test results on IFTTT Dataset. (RT: remove rules shown in training graph).

diverges. The test AUC of **FedRule** increases steadily to the same level as **GraphRule** after 200 iteration. The test Mean Rank also shows similar patterns.

Table 5 shows the final test results of Wyze Smart Home Rules data. Although the **GraphRule** has access to all users' graphs during the training, **FedRule** slightly outperforms **GraphRule** since it employs variance reduction, making the convergence more smooth. **FedRule** greatly outperforms FedGNN in all criteria. The mean rank of **FedRule** is 2.915 after removing the rules in training graphs, which means that for any rules between two specific entities, we need to recommend three rules on average and the user is very likely to adopt at least one of them. Also, Figure 6 shows the test hit rate of the Wyze Smart Home Rules dataset. **FedRule** and **GraphRule** have very close performance and are at most 26.8% better than FedGNN.

Results on IFTTT Dataset. The results of applying these methods on the IFTTT dataset are shown in Figure 5. The IFTTT dataset is relatively smaller than the Wyze Smart Home Rules data, so it is easier to converge during the training process. For the training loss, the centralized training, **GraphRule**, convergences faster since it benefits from the IID distribution of data. For the federated setting, given it is a small dataset, the train loss of FedGNN converges faster than **FedRule** with the same learning rate but the test loss of FedGNN diverges after 50 iterations. Similar to the Wyze
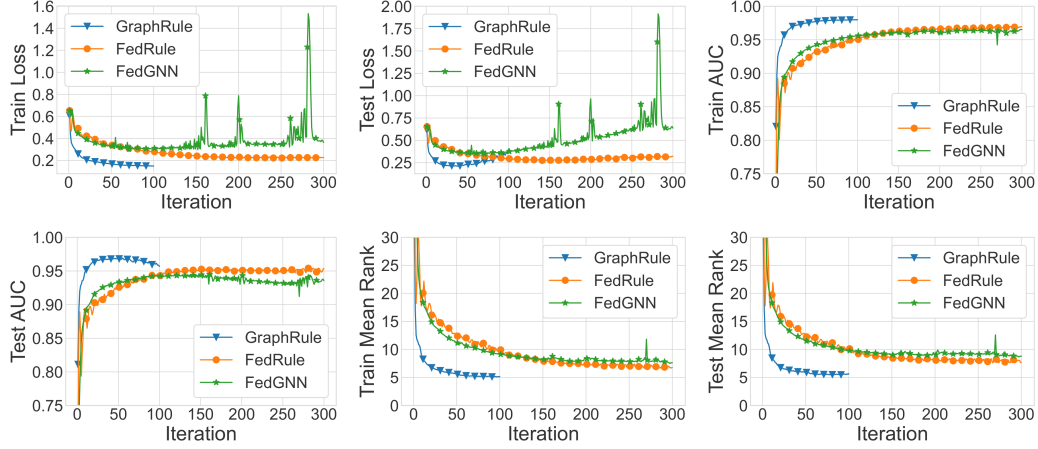
Fig. 8. Training and test performance of different centralized and federated algorithms on Wyze Smart Home Rules Dataset, when considering multiple entities with the same type.

Smart Home Rules dataset, FedGNN has a better test AUC at the start but it decreases after 50 iterations as the test loss diverges. The test AUC of **FedRule** increases steadily during the training. Also, its test mean rank smoothly decreases, converging to an even lower value than the **GraphRule**. Table 6 shows the final test results on IFTTT data. Although **GraphRule** has a better testing loss and a slightly better AUC, the Mean Ranks of **FedRule** are better than those of **GraphRule**, which is a better indicator of the recommendation performance.

Figure 7 shows the hit rate of different algorithms in the IFTTT dataset. Given the huge number of rules and apps, there are trigger-action pairs that are infeasible between some entities in practice. Since the number of users is small and hard to train a general model to avoid these infeasible pairs, we do rule filtering on the recommended rules to keep valid rules. Due to the small size of the dataset, the evaluation has high variance. FedGNN is slightly better than GraphRule and FedRule for the top-5 recommendations. But **FedRule** is the best in general and has at the best 8.1% higher hit rate than **GraphRule** and FedGNN.

Entities with Same Type. As it was mentioned, the graph structure allows us to distinguish between different entities with the same type in a user's graph. In this case, user-item-based methods become infeasible. User graph-based methods, however, can solve this problem by simply considering these entities as different nodes in the graph and using a node embedding to distinguish between different nodes with the same type. Figure 8 shows the training and testing results for centralized and federated methods. Similar to the previous part, **GraphRule** converges faster, and FedGNN diverges after 150 iterations. Similarly, the test loss of FedGNN diverges after 100 iterations while **FedRule** converges smoothly. Also, the AUC and Mean Rank results show that **FedRule** has a better performance than FedGNN.
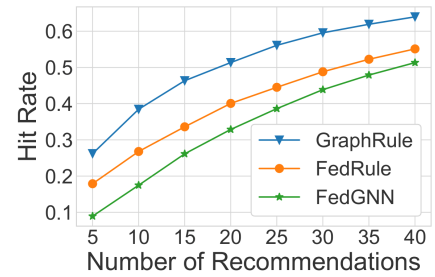

Fig. 9. The Hit Rate on the test set for Smart Home Rules dataset, considering multiple entities with the same type. We compare **GraphRule** with **FedRule** and FedGNN algorithms.

Moreover, Figure 9 shows test hit rates. **GraphRule** has a better performance than federated approaches.
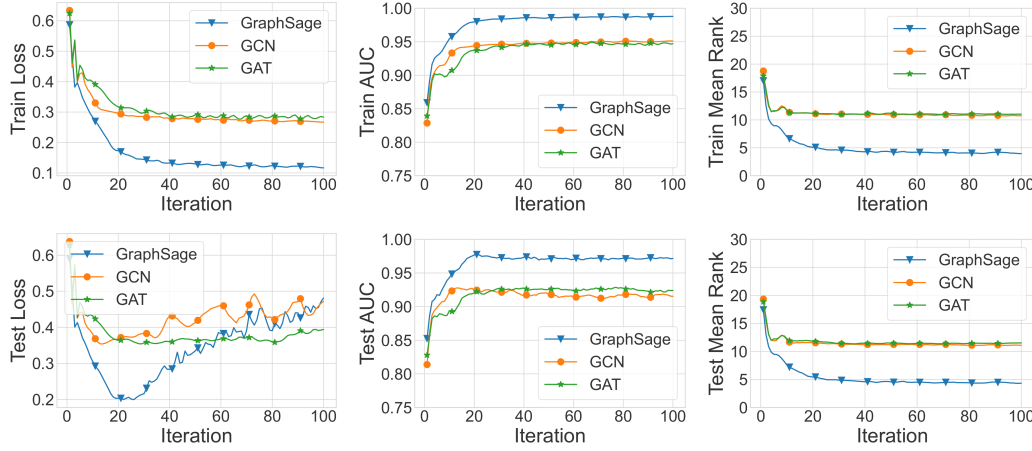
Fig. 10. Train and test performance of different Graph Neural Network models on Wyze Smart Home Rules Dataset. GraphSage has better model performance than GCN and GAT.

4.5 GNN Model Selection

To select the best model for our system we run an ablation study with two other GNN structures. We compare the GraphSage model with GCN [21] and GAT [35]. As shown in Figure 10, GraphSage has better performance (loss, AUC, and mean rank) during training and testing. The main reason is the 0-in-degree nodes in the device graphs. Output for those nodes is invalid since no message will be passed to those nodes, which causes silent performance regression. As in Algorithm 1, for the 0-in-degree nodes, GraphSage concatenates its node feature and the neighbor features (the neighbor feature is a zero vector in this case) to avoid this issue.

4.6 System Implementation

We have implemented our rule recommendation in our smart home system, where users' devices and rules are represented as graphs. The system designed for this purpose has three major components, that are the data pipeline, the training API, and the inference API. For each user, the data pipeline connects to two databases, device ownership, and rules. By combining these two datasets, it can generate a complete graph for each user based on their available rules and devices (even those devices that have not been set up for any rules yet). The data pipeline functionalities are mainly the same for both the training and inference processes. It also generates a 3-dimensional tensor for valid rules based on source and target node types. This tensor will be used to heavily boost the inference time by about 20 times reduction when searching for valid rules. In the training process for the **GraphRule**, we generate the graphs for each user and keep 20% of the rules as the test cases to avoid overfitting. The inference API is written to be called either by each user independently or with a bulk of users in parallel. When the app from the user side calls the inference API, first the data pipeline generates the user graphs, and then the latest trained models are called to run the inference.

Figure 11(a) and 11(b) represent the performance of the API model on a c5.4xlarge instance of Amazon Web Services (AWS). This instance is configured on an Intel® Xeon® Platinum CPU with 3.00GHz frequency and 32 GB of RAM. In Figure 11(a), the time required to complete the inference task from generating the graph to generating new recommendations based on the size of the user's graph is depicted. It is evident that by increasing the graph size the
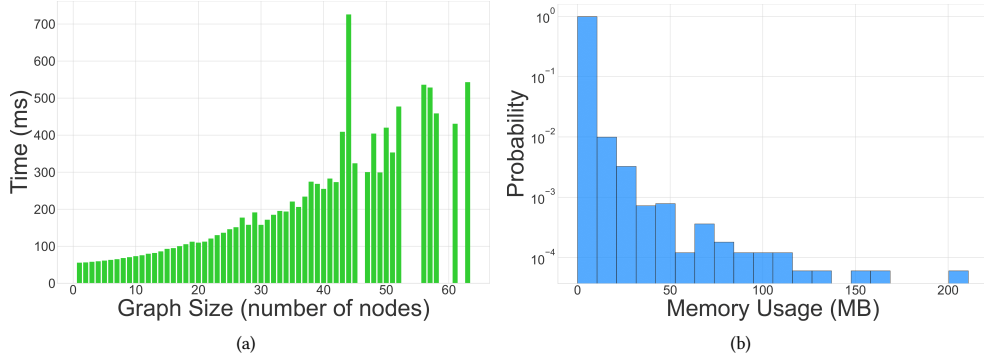
Fig. 11. The system performance of the Inference API for rule recommendation system. Figure (a) shows the delay time of the inference based on the size of the user's graph. Figure (b) shows the histogram of the memory used for the inference for various users. Most of the users got their recommendations with less than 10 MB of memory used and in less than 500 ms.

time required to complete the inference would increase. However, this increase seems to be linear, and for most users, it can be done in less than 500 ms on a normal machine. Figure 11(b), shows the memory usage histogram for different users. As it can be inferred almost all users' inference tasks can be done with as low as 10 MB in memory. This shows that this task can be easily done on edge devices on the user side.

5 CONCLUSION

In this paper, we have presented a new formulation for the rule recommendation systems, called `GraphRule`, based on the graph structure of entities and their connections in a network of users. Despite the user-item format has been widely adopted in conventional recommendation systems, the graph structure is better at handling multiple entities with the same type and providing feasible recommendations based on the available entities in the user's graph. In addition, this graph structure enables us to have local models for each user and hence can further improve the privacy of users. We therefore further propose `FedRule`, a federated rule recommendation setting based on the graph structure. When compared with the previous proposals for federated learning on graphs, `FedRule` is designed for cross-device federated learning to ensure each user's data is kept private locally. Due to the variance reduction mechanisms in `FedRule`, we are able to overcome the Non-IID issue of prior approaches in highly heterogeneous networks of users.

Extensive experimental results on Smart Home Rules and IFTTT datasets present the effectiveness of the proposed formulation and training approaches over prior proposals. Providing more personalized recommendations based on each user's behavior in the federated learning setup is left as a future direction. Another direction of future work might be to learn further about privacy: federated learning models can sometimes be reverse-engineered to reveal private information about a user. To provide a more insightful comparison, real-world implementation of the proposed system and investigating it with rigorous A/B testing is necessary and will be also considered as future work.

ACKNOWLEDGEMENT

REFERENCES

[1] Muhammad Ammad-Ud-Din, Elena Ivannikova, Suleiman A Khan, Were Oyomno, Qiang Fu, Kuan Eeik Tan, and Adrian Flanagan. 2019. Federated collaborative filtering for privacy-preserving personalized recommendation system. *arXiv preprint arXiv:1901.09888* (2019).

[2] James Bennett, Stan Lanning, et al. 2007. The netflix prize. In *Proceedings of KDD cup and workshop*, Vol. 2007. New York, NY, USA., 35.

[3] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263* (2017).

[4] Dheeraj Bokde, Sheetal Girase, and Debajyoti Mukhopadhyay. 2015. Matrix factorization model in collaborative filtering algorithms: A survey. *Procedia Computer Science* 49 (2015), 136–146.

[5] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. 2019. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046* (2019).

[6] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.

[7] Di Chai, Leye Wang, Kai Chen, and Qiang Yang. 2020. Secure federated matrix factorization. *IEEE Intelligent Systems* (2020).

[8] Mingyang Chen, Wen Zhang, Zonggang Yuan, Yantao Jia, and Huajun Chen. 2020. Fede: Embedding knowledge graphs in federated setting. *arXiv preprint arXiv:2010.12882* (2020).

[9] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. 2019. RecRules: recommending IF-THEN rules for end-user development. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 5 (2019), 1–27.

[10] Yuyang Deng, Mohammad Mahdi Kamani, and Mehrdad Mahdavi. 2020. Adaptive personalized federated learning. *arXiv preprint arXiv:2003.13461* (2020).

[11] Yuyang Deng, Mohammad Mahdi Kamani, and Mehrdad Mahdavi. 2020. Distributionally robust federated averaging. *Advances in Neural Information Processing Systems* 33 (2020), 15111–15122.

[12] Rainer Gemulla, Erik Nijkamp, Peter J Haas, and Yannis Sismanis. 2011. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 69–77.

[13] Carlos A Gomez-Uribe and Neil Hunt. 2015. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)* 6, 4 (2015), 1–19.

[14] Farzin Haddadpour, Mohammad Mahdi Kamani, Mehrdad Mahdavi, and Viveck Cadambe. 2019. Local sgd with periodic averaging: Tighter analysis and adaptive synchronization. *Advances in Neural Information Processing Systems* 32 (2019).

[15] Farzin Haddadpour, Mohammad Mahdi Kamani, Aryan Mokhtari, and Mehrdad Mahdavi. 2021. Federated learning with compression: Unified analysis and sharp guarantees. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2350–2358.

[16] Farzin Haddadpour and Mehrdad Mahdavi. 2019. On the convergence of local descent methods in federated learning. *arXiv preprint arXiv:1910.14425* (2019).

[17] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 1025–1035.

[18] Chaoyang He, Keshav Balasubramanian, Emir Ceyani, Yu Rong, Peilin Zhao, Junzhou Huang, Murali Annavaram, and Salman Avestimehr. 2021. Fedgraphnn: A federated learning system and benchmark for graph neural networks. *arXiv preprint arXiv:2104.07145* (2021).

[19] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).

[20] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. 2020. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*. PMLR, 5132–5143.

[21] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[22] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[23] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems* 2 (2020), 429–450.

[24] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. 2019. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189* (2019).

[25] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7, 1 (2003), 76–80.

[26] Chao Liu, Hung-chih Yang, Jinliang Fan, Li-Wei He, and Yi-Min Wang. 2010. Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce. In *Proceedings of the 19th international conference on World wide web*. 681–690.

[27] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.

[28] Xianghang Mi, Feng Qian, Ying Zhang, and XiaoFeng Wang. 2017. An empirical characterization of IFTTT: ecosystem, usage, and performance. In *Proceedings of the 2017 Internet Measurement Conference*. 398–404.

[29] Arvind Narayanan and Vitaly Shmatikov. 2006. How to break anonymity of the netflix prize dataset. *arXiv preprint cs/0610105* (2006).

[30] Hao Peng, Haoran Li, Yangqiu Song, Vincent Zheng, and Jianxin Li. 2021. Federated Knowledge Graphs Embedding. *arXiv preprint arXiv:2105.07615* (2021).

[31] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks* 20, 1 (2008), 61–80.

[32] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*. Springer, 593–607.

[33] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197* (2019).

[34] Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze, and Michael L Littman. 2016. Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 3227–3231.

[35] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[36] Frank Edward Walter, Stefano Battiston, and Frank Schweitzer. 2008. A model of a trust-based recommendation system on a social network. *Autonomous Agents and Multi-Agent Systems* 16, 1 (2008), 57–74.

[37] Chuhan Wu, Fangzhao Wu, Yang Cao, Yongfeng Huang, and Xing Xie. 2021. Fedgnn: Federated graph neural network for privacy-preserving recommendation. *arXiv preprint arXiv:2102.04925* (2021).

[38] Qinyue Wu, Beijun Shen, and Yuting Chen. 2020. Learning to Recommend Trigger-Action Rules for End-User Development. In *International Conference on Software and Software Reuse*. Springer, 190–207.

[39] Yuhang Yao and Carlee Joe-Wong. 2021. Interpretable clustering on dynamic graphs with recurrent graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 4608–4616.

[40] Yuhang Yao and Carlee Joe-Wong. 2022. Fedgcn: Convergence and communication tradeoffs in federated training of graph convolutional networks. *arXiv preprint arXiv:2201.12433* (2022).

[41] Tianlong Yu, Tian Li, Yuqiong Sun, Susanta Nanda, Virginia Smith, Vyas Sekar, and Srinivasan Seshan. 2020. Learning context-aware policies from multiple smart homes via federated multi-task learning. In *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 104–115.

[42] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 793–803.

[43] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems* 31 (2018), 5165–5175.

[44] Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, and Can Wang. 2019. Hierarchical graph pooling with structure learning. *arXiv preprint arXiv:1911.05954* (2019).

[45] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. 2018. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582* (2018).