Predicting Category of Next Point of Interest (POI)

Dobromir M. Marinov
Department of Computer Science
University of Cambridge
Cambridge, CB3 0FD
dmm80@cam.ac.uk

Jacky W. E. Kung
Department of Computer Science
University of Cambridge
Cambridge, CB3 0FD
jwek2@cam.ac.uk

Abstract

Recommending POIs have had a vital role in enhancing the user experience within Location-Based Social Networks (LBSNs). Traditional machine learning methods usually gather all user data in a centralised fashion, in order to train their models, with little regard for the individual users' privacy. In this work we explore a federated learning approach to building a model that allows user data to never leave individual devices by only sharing model updates. We conduct a number of experiments covering 3 different scenarios. First, we explore if the proposed federated model performs better than models trained on individual devices. Second, we explore the effect that data quantity skew has on the model. Third, we test the resilience of the federated learning procedure towards client drop-outs. Our extensive experiments demonstrate that federated learning-based approaches offer a good performance without the need to forgo personal data privacy.

1 Introduction

With the increase in popularity of Location-Based Social Networks (LBSNs), such as Foursquare, it has become easier to collect large amounts of Points-of-Interest (POIs) data which represents users visiting different venues, such as cafes, restaurants and shops. The large amounts of data that is generated on LBSNs has been shown to be crucial for building accurate recommendation systems using deep learning-based methods [1]. However, this type of user data is inherently sensitive and it is therefore important to ensure a high degree of privacy, when training deep learning models on it, while also trying to avoid any potential performance decreases in their predictive power. One popular approach to training machine learning models without compromising the privacy of user data is the adoption of federated learning techniques. It has been shown that federated learning methods can build models that achieve recommendation quality comparable to models trained using centralised methods, while maintaining a high degree of data privacy [2] by keeping all sensitive data on the user's devices.

In this work, we explore two different machine learning methods for predicting the category of the next POI and extend and adapt them to a federated learning context using the Flower framework [3]. For each of the two methods we conduct a base experiment to test the hypothesis that a model trained in a federated scenario would outperform models trained on individual clients. We conduct further experiments to quantify the effects that different levels of data quantity skew can have on the federated learning model. Finally, we construct a simulated scenario in which we train a federated model with unreliable clients that can drop-out during training and explore and discuss the effects on the performance of the federated model for the different drop-out rates. Our experimental results show that while there are specific challenges that come with adopting a federated learning approach, it is a viable strategy that can achieve adequate performance, while protecting user privacy. We finish the report by addressing the aforementioned challenges and outlining promising directions for future work.

Preprint. Under review.

2 Background

Islam et al. provided an excellent overview of deep learning approaches to POI recommendation [1]. A plethora of methods have been employed to this task using data from LBSNs such as Foursquare.As the POI recommendation problem is typically modeled as the prediction of the next POI given a historical sequence of POIs, neural network models that accept sequential data, such as Recurrent Neural Networks (RNNs), are a promising first approach to the problem.

However, a central problem hindering the efficacy of vanilla RNNs is the sparsity of POI check-in data. While there is strong academic evidence that regular and periodic spatiotemporal patterns exist in POI traces [4], people may not perform LBSN check-ins with the same regularity. Check-ins in datasets are therefore typically separated by large and irregular spatiotemporal gaps, hindering the learning of patterns [1, 5, 6, 7]. Therefore, RNNs are typically augmented with temporal and spatial distances of historical POIs with respect to the current time and location of a user [5, §1]. Liu et al. proposed ST-RNN in 2016, in which the RNN explicitly takes these as additional inputs [8], and became a state-of-the-art solution at the time. As an improvement to ST-RNNs, Yang et al. later proposed Flashback in 2020, in which the spatiotemporal factors are used not as explicit inputs to RNN cells, but as weighting factors for historical RNN outputs, to better capture the diurnal periodicity of real-life POI traces. We explore its technical details in Section 3.2.

Over the past decade, RNNs have been superseded by derivative architectures. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) models can retain information for much longer than RNNs, overcoming the "vanishing gradient problem". Better-performing models today mostly involve transformer architectures to selectively pay attention to specific parts of earlier inputs in the input sequence [9]. Separately, graph embedding models have also been effective, by representing spatiotemporal interactions in latent spaces trained for next POI recommendation.

Today, privacy-preserving recommendation systems are of significant interest. Federated learning techniques are popular, as they allow sensitive data such as POI histories to remain on users' devices, while still training a federated model via the exchange of minimal information (e.g. model parameters) with a coordinating server [10]. Sun et al. [7] provide a comprehensive overview of federated recommendation systems (not limited to POI recommendations). Advanced setups apply differential privacy and secure aggregation techniques to achieve provable privacy guarantees [11, 12].

Federated learning is characteristically plagued by the challenges of data heterogeneity and device heterogeneity. Because data is not held centrally, standard deep learning techniques to ensure that data is independently and identically distributed cannot be applied, which can result in the biasing of results away from optimal, and unfairness in model performance towards clients with large datasets (quantity skew). Furthermore, different client devices have different processing speeds and connectivity characteristics, and may result in client drop-outs or stragglers during federated learning. We investigate the latter two effects in this work.

Several federated learning approaches for POI recommendation exist. In PREFER, instead of using a single global coordinating server, which presents a single point of failure, numerous edge servers are spread across the world, each aggregating model updates from local communities of users [13]. It provides good personalisation of POI recommendations to local users while curbing overall system complexity, though computation time on user devices remains significant and could theoretically be improved [13, §5.2.2]. In FL&PP-POI, homomorphic encryption is used, allowing operations on encrypted data as if they were decrypted [14, §IIIC]. It resolves the cold start problem in new cities (where data is absent) by leveraging existing data from auxiliary domains such as e-commerce, leading to consistent performance gains. However, empirical evaluation of model efficiency was omitted, hence computational costs are potentially prohibitively expensive. In FedPOIRec, homomorphic encryption is also used, for secure aggregation [15, §2.3]. It also supports tailoring of recommendations to social circles, but uses complicated models that have non-trivial communication and storage costs (dominated by cryptographic keys used in homographic encryption) [15, §6.4].

While the challenges of federated learning are known, their analysis specific to the context of POI recommendation is absent in the literature, to the best of our knowledge. Modern federated POI recommendation methods tend to use complicated system and model architectures, which do not lend well to quick prototyping. To address the literature gap, we adapt small model architectures from the centralised setting to the federated setting. While we were unfortunately unable to obtain pre-trained models, we found two efficient open-source models, which we describe in the next section.

3 Methods

3.1 MCMG

One of the methods for predicting next point-of-interest (POI) that we have chosen to adapt to the federated learning context is the Multi-Channel next POI recommendation framework with Multi-Granularity check-in signals (MCMG) developed by Sun et al. [6]. The model that they propose combines three separate modules, a global user behaviour encoder, a local multi-channel encoder and a region-aware weighting strategy. The purpose of the global user behaviour encoder is to capture the overall sequential patterns in the global check-in data. This is achieved by first building a list of trajectories for each user, where each trajectory represents a chronologically ordered sequence of check-ins for that user for a single day. The user trajectories are then used in order to construct a directed graph in which each POI from the trajectories is represented as a node and each pair of points from the trajectories are linked by a directed edge. Finally, a graph convolutional network (GCN) is trained on the trajectories in order to learn the popular sequential preferences across all users. The GCN is defined as:

$$\mathbf{H}^{(z+1)} = ReLu(\tilde{\mathbf{D}}^{-1}\mathbf{A}\mathbf{H}^{(z)}\mathbf{W}^{(z)}) \tag{1}$$

where the sum of the adjacency matrix and identity matrix is detonated as $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, with $\mathbf{A} \in \mathbb{R}^{|L|\times|L|}$. The diagonal in-degree matrix is detonated as $\tilde{\mathbf{D}} \in \mathbb{R}^{|L|\times|L|}$ with $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$. The POI embedding matrix in the z-th layer is denoted as $\mathbf{H}^{(z)} \in \mathbb{R}^{|L|\times d}$, with d denoting the embedding size and the layer-wise trainable weight matrix is denoted as $W^{(z)} \in \mathbb{R}^{d\times d}$. The second module, the local multi-channel encoder is utilised in order to enable the global model to learn personalised transition patterns of individual users based on the location, region and activity data from their POI check-ins. This module defines three different encoders, a location encoder, a region encoder and a category encoder, which are referred to as "channels". The encoders of each channel comprise a GCN that uses the POI trajectories in order to learn the user-specific behaviour for each channel. This is important because it allows the MCMG model to capture the spatiotemporal patterns of the users' activities and enables context-aware correlations for non-consecutive check-ins [6].

Finally, the third module, the region-aware weighting strategy, is used to aggregate the signals from the previously described channels in order to produce a more accurate POI prediction. This is achieved by using a region-aware weighting strategy that dynamically adapts the impact of different channels. The core idea behind it is to split the user trajectories into two separate groups, based on whether or not all of the points of interest, that they consist of, are from the same region or from multiple different regions. The channels corresponding to the same-region group are then assigned higher weights, due to the higher likelihood of producing more reliable predictions while the channels corresponding to the cross-region group are weighted lower. The final aggregated representation of the user preference is then formulated as a sum of the products of the learnable weights multiplied by the corresponding embeddings of the POIs for each of the previously described channels. While we consider the above explanation of MCMG to be sufficient for the purposes of our work, a more thorough explanation of the method and justifications for its assumptions can be found in the original work by Sun et al. [6].

The original code with our modifications for running the baseline MCMG model is open-sourced on GitHub.[1]

3.2 Flashback

The second model we chose for predicting next POI is Flashback developed by Yang et al. in 2020 [5]. It builds on top of recurrent neural networks (RNNs). Let U, P represent the set of all known users and POIs respectively. Each user $u \in U$ is represented by a user embedding $\mathbf{u} \in \mathbb{R}^{10}$; each POI $p_i \in P$ at position i in a sequence is represented by a POI embedding $\mathbf{p}_i \in \mathbb{R}^{10}$, and occurred at time t_i at latitude and longitude $\mathbf{x}_i = [x_i \quad y_i]^T$. At the core of Flashback, an RNN unit $\mathcal{F}$ takes in the current POI $\mathbf{p}_i$ and the previous hidden state $\mathbf{h}_i \in \mathbb{R}^{10}$, outputs a context vector $\mathbf{c}_i \in \mathbb{R}^{10}$ and the next hidden state $\mathbf{h}_{i+1}$:

$$\mathcal{F}(\mathbf{p}_i, \mathbf{h}_i) = (\mathbf{c}_i, \mathbf{h}_{i+1}). \tag{2}$$

[1] `https://github.com/DobromirM/MCMG`

This RNN unit then takes in successive POIs $\in \{\mathbf{p}_0, \ldots, \mathbf{p}_{T-1}\}$. In order to predict $\mathbf{p}_T$, Flashback first forms a weighted average of the previous K (up to a fixed limit, set to 20 [5, §4.4]) context vectors $\mathbf{c}_i$ using a weighting function f_w, then combines it with the user's embedding $\mathbf{u}$, before passing them through a fully-connected layer $\mathcal{G}$ to produce a logit vector $\mathbf{y} \in \mathbb{R}^{|P|}$ (probabilities that are trained to sum to 1). $\mathbf{p}_T$ is then taken to correspond to the largest logit element,

$$\mathbf{y} = \mathcal{G}\left(\mathbf{u}, \frac{1}{Z}\sum_{k=1}^{\min(K,T)} f_w\Big(t_T - t_{T-k}, \|\mathbf{x}_T - \mathbf{x}_{T-k}\|_2\Big) \cdot \mathbf{c}_{T-k}\right) = \begin{bmatrix} y_0 & \cdots & y_{|P|-1} \end{bmatrix}^T, \quad (3)$$

$$\mathbf{p}_T = \operatorname{argmax}_j y_j, \quad (4)$$

where Z is a normalising factor to get the weights to sum to 1, and $\|\cdot\|_2$ is the L2-norm. The context vector weighting function f_w exponentially penalises context vectors from historical POIs that are further away in time and space, while also sinusoidally rewarding context vectors at a daily frequency counting back from the current time, following strong academic evidence of diurnal patterns in POI traces [5, §1]. For further mathematical treatment, we refer to the original paper [5, §3.2]. While Flashback supports RNN derivatives such as Long Short-Term Memory (LSTMs) and Gated Recurrent Units (GRUs), basic RNNs were found to perform the best [5, Table 2]. We thus focus on Flashback RNN, which uses only a basic RNN for $\mathcal{F}$, in the remainder of this work.

4 Experimental setup and methodology

4.1 Adaptation to Federated Learning

We adapt both MCMG and Flashback to the federated learning setting using the Flower framework [3], following the provided template.[2] The full lists of hyperparameters that we have used for training the MCMG and Flashback models are provided in Tables A11–A12. However, as they were designed for centralised learning, adaptation posed challenges.

The original implementation of MCMG is trained for 100 epochs with a "patience" hyperparameter that stops the training procedure early in the cases where the loss is not improving for a certain number of rounds. In the federated learning context, we train each client's model with the same epoch strategy as described above. Furthermore, our choice of hyperparameters for the MCMG models is based on the extensive hyperparameter search experiment that has been conducted in the original paper by Sun et al. [6].

Flashback, as originally written, is trained over 100 epochs using a `MultiStepLR` scheduler to decrease the learning rate every 20 epochs. In Flower, we train for 20 rounds, each lasting 5 epochs. However, because the template does not pass the server round number to `train()` by default (needed for `MultiStepLR`), small modifications to the template were made. More importantly, the presence of trainable user-specific embeddings (`nn.Embedding` modules) complicated matters. As they can identify users, these modules should be user-private and not seen by other clients. However, Flower's strategies (e.g. `FedAvg`) pass all model parameters from the server to active clients in each round. To simulate a more realistic privacy-preserving setup, a custom strategy was subclassed from `FedAvg`, to pass only the one user embedding module relevant to each active client, and handle proper aggregation of these lone embedding modules into a persistent server store at the end of each round.

The implementation of our code for running the federated learning experiments, described in the following sections, on both MCMG and Flashback, are open-sourced on GitHub.[3]

4.2 Datasets

We use a subset of the Foursquare dataset [16], which contains check-in records between April 2012 and September 2013 for two cities, Calgary (CAL) and Phoenix (PHO) (see Table A4). The datasets have been extended by Sun et al. by splitting the data for each city into 9 geographical regions and assigning each data point to a specific region [6]. This is done by clustering the data points using a k-means algorithm. We use these datasets as the starting point for both MCMG and Flashback;

[2] `https://github.com/camlsys/fl-project-template`

[3] `https://github.com/jackykwe/l361-project-next-poi-prediction`

full statistics of the raw datasets can be found in A4. Additional data processing procedures were performed, as described in the following sections. While the original work on MCMG includes data for two additional cities, we have chosen not to use the additional data as it will not contribute significantly to testing our core hypotheses and its larger size will require more compute resources.

4.2.1 MCMG data pre-processing

The raw check-in records are used to build user trajectories where each user trajectory is a sequence of chronologically ordered visits for a single user in a single day. All users that contain less than three trajectories are filtered from the pre-processed dataset and for all other users, the first 80% of their trajectories are used as the training set, the next 10% are used as the test set and the last 10% are left as a validation set. In addition to the POI trajectories, for each user, a number of supplementary trajectories are generated. These supplementary trajectories correspond to additional POIs attributes, such as the category of the visited venue, the region to which the POI belongs, the distance between the POIs and the time of day of the visit. The distance and time of day trajectories are derived from the original data as the delta of the locations and the times of consecutive visits, respectively. They follow the same format as the POI trajectories and are represented as chronologically ordered values over a single day and are used as part of the training procedure in MCMG. It should be noted that in the original implementation of MCMG consecutive visits to the same location by a single user are also collapsed into a single visit. We adopt this strategy in order to stay consistent with the original work.

4.2.2 Flashback data pre-processing

Following the original work, users with fewer than 101 check-ins are ignored; this is to ensure that following an 80-20 train-test split, there exists at least 20 check-ins in both the train and test sets to form one length-20 "sequence", which is the granularity at which the Flashback model processes check-ins. Check-ins $\{p_i\}_{i=0}^{N}$ (with $N \geq 101$) are sorted chronologically, then an off-by-one alignment is done to align historical POIs $\mathbf{x}$ with corresponding next POIs $\mathbf{y}$:

$$\mathbf{x}, \mathbf{y} = \{p_i\}_{i=0}^{N-2}, \{p_i\}_{i=1}^{N-1}. \tag{5}$$

These length-$(N-1)$ $\mathbf{x}, \mathbf{y}$ are then split into train and test sets in an $80:20$ ratio, and length-20 sequences are formed by taking check-ins 20 at a time from the beginning of each split, dropping any remainder check-ins that do not form a full sequence. During evaluation, each test sequence corresponds to 20 predictions: for each of $k \in \{0, \ldots, 19\}$, predict p_{k+1} given $\{p_i\}_{i=0}^{k}$. The procedure here is exactly as implemented in the original work. Finally, only five attributes from the Foursquare dataset are involved: user ID, venue ID, timestamp, latitude and longitude.

4.3 Performance metrics

In order to measure and compare the performance of the POI recommender models we adopt a popular metric [17]: Hit Ratio at rank K (HR@K) [5, 6] (also known in literature as Recall@K). The HR@K metric is defined as the percentage of users for whom at least one relevant item appears within the top K recommendations generated by the system. In our experiments, instead of taking a simple percentage (which corresponds to unweighted averaging), we perform a weighted average over all users, weighted based on their local dataset sizes (i.e. number of check-ins of each user). This interpretation of HR@K was specifically chosen as Flower returns by default only weighted averaged metrics, without unweighted averages or the raw values of each client before aggregation. While in our repository we measure $K \in \{1, 5, 10\}$, we only report $K \in \{1, 5\}$ because some experimental setups resulted in missing values for $K = 10$ (this does not affect our discussions). We focus on predicting the category of the next point of interest for users in all of our experiments.

4.4 Hypotheses and experiments

All experiments are run thrice with different random number generator seeds. During federated training, we sample $\approx 10\%$ of all eligible clients. We chose the most basic aggregation strategy: Flower's implementation of `FedAvg`; exploration of more advanced strategies is outside the scope of this work. Measures taken to mitigate potential sources of error are deferred to Section 5.4.

We now formulate as hypotheses three challenges of federated learning that we investigate in this work.

H1 POI recommendation performance improves when shifting from non-federated single-client training to federated training.

Federated training is performed via Flower. Single-client training is performed outside of Flower for MCMG using our modifications at Footnote 1,

Concretely, the weighted average metrics (§4.3) after all training terminates should be superior in the federated training setting, compared to the single-client case where the models are trained locally using strictly only the data on each client.

H2 POI recommendation performance for federated training worsens with increased severity of dataset quantity skew among clients.

We consider three levels of quantity skew: *Very heterogeneous*, *slightly heterogeneous* and *approximately homogeneous*. The first case is simply the full pre-processed dataset without further modifications. The slightly heterogeneous case is obtained by considering the dataset size distribution across all clients in a given city, then retaining only clients whose dataset size is within one standard deviation of the mean. The approximately homogeneous case is obtained by retaining only the 15% of all clients closest to the mode of the distribution.

Details characteristics of these reduced datasets are available in Appendix B.2.

H3 POI recommendation performance for federated training worsens with increased device heterogeneity.

In particular, we study the effect of client dropouts, and consider five dropout probabilities $\in \{0, 0.1, 0.15, 0.2, 0.25\}$. At training time, each client has a preset probability of dropping out (for any of a plethora of reasons, including intermittent connectivity and device running low on battery). If dropped out, the model returns from the `train()` method, passing a dataset size of 0 back to the server, so they are not involved in the aggregation for the particular round.[4]

Similarly to H1, we compare weighted average metrics across the different scenarios for H2 and H3.

5 Results and discussion

5.1 Single-client training versus federated training (H1)

In the first set of experiments for the MCMG algorithm we compare the performance of a federated model, evaluated on each of the clients' test datasets against the performance of client models trained only on the data from their respective devices and evaluated on their own test datasets. We run the experiments 3 times in order to obtain minimum, maximum and median values. The results on both the CAL and PHO datasets, summarised in Table 1, confirm H1 by showing that the federated model consistently outperforms single-client models. This is expected as the federated model is exposed to a much higher quantity of data, even if it is done indirectly – by aggregating the updates of models trained on the user devices.

On the other hand, experiments using the Flashback algorithm surprisingly showed the direct opposite result: single-client training outperforms federated training, and by a large margin too (Table 1). Re-runs of the experiment using different seeds did not change the result. We conjecture that this is due to over-fitting in the single-client case, and under-fitting in the federated case. From Table A8, there are 472 unique locations in the CAL dataset. However, from Figure 1, even the client with the largest local dataset had only 80 unique locations (17% of total). With a greatly reduced number of locations to consider at each client, their location embeddings module is very likely to have over-fitted, as evidenced by the training loss plateau in Figure 1. On the other hand, with all 472 locations to consider, it would seem that the federated Flashback model did not converge. This is likely due to poorly tuned hyperparameters; we use the defaults provided by the Flashback paper for the Foursquare dataset [5, §4.3], but as this work used MCMG's rendition of the Foursquare dataset

[4] If all clients in a round drop out, there would be a division-by-zero error at the aggregation stage (during weighted averaging). We wanted to return a small float (e.g. `1e-9`), but that led to further crashes. We thus return the smallest non-zero integer `1` in our implementation.

Table 1: Comparison between the performance of single-client and federated settings of MCMG and Flashback models trained on the CAL and PHO datasets. All values are weighted averaged across all clients in the dataset, weighted by local dataset size.

Model	Dataset	Setting	HR@1			HR@5		
			Min	Max	Median	Min	Max	Median
MCMG	CAL	Single-client	0.0023	0.0043	0.0032	0.0113	0.0172	0.0117
		Federated	**0.0374**	**0.0510**	**0.0451**	**0.0909**	**0.1441**	**0.1101**
	PHO	Single-client	0.0014	0.0019	0.0017	0.0060	0.0067	0.0066
		Federated	**0.0130**	**0.0212**	**0.0175**	**0.0261**	**0.0588**	**0.0332**
Flashback	CAL	Single-client	**0.2052**	**0.2156**	**0.2094**	**0.4812**	**0.5000**	**0.4812**
		Federated	0.0333	0.0406	0.0344	0.1271	0.1271	0.1271
	PHO	Single-client	**0.2171**	**0.2319**	**0.2196**	**0.4966**	**0.5053**	**0.5042**
		Federated	0.0199	0.0277	0.0260	0.0676	0.0696	0.0686

Figure 1: Left: plot of training loss against round number for Flashback. Right: distribution of number of unique locations in each client's local dataset, for the single-client case. These are for the CAL dataset.

instead (for consistency between the two models), prior hyperparameter re-tuning may have been necessary for Flashback to achieve convergence here. We leave hyperparameter re-tuning as future work. The same argument applies for the PHO dataset, which showed similar results and is thus not shown.

5.2 Effects of different quantity skew on federated training (H2)

In the second set of experiments for the MCMG algorithm we compare the performance of the federated model between the three levels of client dataset quantity skew described in H2. The results, summarised in Table 2, are surprising, at first glance, due to the homogeneous case achieving lower performance compared to the heterogeneous cases on both dataset. However, if we look at the number of check-ins after the data is pre-processed, summarised in Table A5, we can see that the homogeneous case has a tiny number of data points. This is mainly caused by the way that the MCMG trajectories are created and shows that for smaller datasets, trying to homogenise the dataset will most likely lower the performance, instead of increasing it. Due to limits in both the computational resources and time that we have available, we could not perform the full experiment on larger datasets and will leave that as a direction for future work.

On the other hand, for Flashback, the results were uniform across Table 2, confirming our hypothesis by showing that the approximately homogeneous case consistently outperforms the more heterogeneous cases. This is expected: in a highly heterogeneous case, the weighted average metrics would be dominated by clients with unusually large dataset sizes (on the right of the graphs in Figure 2. With our low per-round client sampling rate of $\approx 10\%$, there is a good chance that these outlier clients would be missed, which would severely push down the weighted average metrics. This effect grants homogeneous subsets of an original dataset an edge over heterogeneous subsets.

Table 2: Comparison between different levels of data quantity skew of federated MCMG and Flashback models trained on the CAL and PHO datasets. All values are weighted averaged across all clients in the dataset, weighted by local dataset size.

Model	Dataset	Data quantity skew	HR@1			HR@5		
			Min	Max	Median	Min	Max	Median
MCMG	CAL	Very heterogeneous	0.0230	0.0460	0.0351	**0.0908**	**0.1488**	**0.1339**
		Slightly heterogeneous	**0.0351**	**0.0540**	**0.0379**	0.0879	0.1258	0.1190
		Approximately homogeneous	0.0040	0.0149	0.0149	0.0230	0.0581	0.0339
	PHO	Very heterogeneous	0.0086	0.0184	0.0098	**0.0312**	**0.1221**	0.0436
		Slightly heterogeneous	0.0110	**0.0310**	**0.0310**	0.0310	0.0846	**0.0846**
		Approximately homogeneous	**0.0157**	0.0224	0.0224	0.0303	0.0665	0.0386
Flashback	CAL	Very heterogeneous	0.0333	0.0406	0.0343	0.1271	0.1271	0.1271
		Slightly heterogeneous	0.0167	0.0179	0.0167	0.1038	0.1090	0.1077
		Approximately homogeneous	**0.0900**	**0.1200**	**0.1000**	**0.2700**	**0.2900**	**0.2800**
	PHO	Very heterogeneous	0.0199	0.0277	0.0260	0.0676	0.0696	0.0686
		Slightly heterogeneous	0.0155	0.0159	0.0159	0.0420	0.0425	0.0425
		Approximately homogeneous	**0.1179**	**0.1179**	**0.1179**	**0.1857**	**0.1964**	**0.1929**

5.3 Effects of different dropout probabilities on federated training (H3)

In the third set of experiments for the MCMG algorithm we compare the performance of the federated model between the five drop-out rates described in H3. The results, summarised in Table 3, contrary to our expectations, show no clear correlation between lower drop-out rates and higher model performance. However, it is important to note that, despite the fact that the models trained with the lowest drop-out rate do not achieve the highest model performance, it is clear that they mostly outperform the models with the highest drop-out rate. One possible explanation for the similarity between the performances of models trained with different levels of drop-out is that due to the small number of trajectories per client that MCMG trains on, it is possible that single clients do not contribute significantly to the overall performance of the final model. In addition, the relatively low drop-out rates, coupled with the small number of clients per round create only small differences between the different experimental cases, which become more pronounced only as the drop-out rate becomes higher, as supported by the experimental results. Due to the fact that the datasets we have used only contain clients with small number of data points, it is infeasible to train federated models with higher drop-out rates. As part of future work, it will be interesting to test our assumptions using a larger datasets and higher drop-out rates.

Similarly surprisingly, for Flashback, the results in Table 3 showed zero variation across the five drop-out rates. We conjecture that, given that the chosen drop-out rates are rather low, there is a good chance that the random number generator (using the chosen seeds) always produces a float $\in [0, 1)$ greater than 0.25, resulting in no differences in drop-out behaviour. Other experimental runs involving different sets of three seeds were performed, but no differences between the five were ever observed. We suspect we have picked sets of "lucky seed combinations"; further testing involving more seed combinations is required before making conclusions. Furthermore, there could also be more complicated interactions between the random sources involved in determining client drop-out and client selection: it could be that there were differences in client drop-outs between the five, but the drop-outs ended up having no effect on the final evaluation score. Other effects such as label distribution homogeneity between clients could also dampen the effect of drop outs. Deeper investigations are required before definitive conclusions can be drawn.

5.4 Potential sources of error and mitigations

MCMG's Foursquare dataset. While using the same raw dataset for all our experiments provides consistency, it also becomes a root source of error. The best we could do is to pre-process it following MCMG's and Flashback's implementations as closely as we could; biases or inaccuracies within the raw dataset are outside of our control.

Table 3: Comparison between different levels of client drop-out rates of federated MCMG and Flashback models trained on the CAL and PHO datasets. All values are weighted averaged across all clients in the dataset, weighted by local dataset size.

Model	Dataset	Drop-out rate	HR@1			HR@5		
			Min	Max	Median	Min	Max	Median
MCMG	CAL	0.00	0.0000	0.0972	0.0341	**0.1000**	0.2115	**0.1968**
		0.10	**0.0462**	**0.1154**	**0.0500**	0.0500	0.2308	0.0914
		0.15	0.0341	0.0612	0.0370	**0.1000**	**0.3846**	0.1164
		0.20	0.0000	0.0521	0.0357	0.0357	0.1945	0.1072
		0.25	0.0244	0.0357	0.0300	0.0357	0.2167	0.0911
	PHO	0.00	0.0185	0.0260	0.0196	0.0185	**0.1316**	0.0747
		0.10	**0.0196**	0.0453	**0.0260**	**0.0647**	0.1119	0.0768
		0.15	0.0000	0.0196	0.0163	0.0453	0.1170	**0.1119**
		0.20	0.0000	**0.0493**	0.0181	0.0000	0.1229	0.0380
		0.25	0.0000	0.0453	0.0235	0.0000	0.0747	0.0453
Flashback	CAL	*all five*	0.0333	0.0406	0.0344	0.1271	0.1271	0.1271
	PHO	*all five*	0.0199	0.0277	0.0260	0.0676	0.0696	0.0686

Implementation errors. While both MCMG and Flashback were once state-of-the-art, we achieved inconsistent results when adapting them to the federated setting. This could be due to implementation faults on our part. To reduce the likelihood of such faults, we strived to re-use as much code as possible from the original implementations, tweaking only what is necessary to get them to work in Flower. To overcome our unfamiliarity with the Flower and Hydra libraries, significant efforts were spent tracing function calls to understand how its components fit together.

Stroke of luck. Where experimental results are surprising, we perform further runs using more seeds to determine if those surprises are one-off effects caused by "lucky seeds". This is generally achieved by running all experiments three times using three different seeds (Table A13). However, time and computing power limits exist on the number of additional experiments we could run.

6 Conclusions and future work

In conclusion, we have successfully adapted two classical machine learning algorithms for predicting the category of the next point of interest to a cross-device federated learning scenario. Through experimental work, we have shown that the federated learning models perform better than their individual-device counterparts. We have also provided a comprehensive discussion of selected forms of data and device heterogeneities, contextualised to POI recommendation systems. Specifically, we have quantified the effects of data quantity skew and client drop-out on the MCMG and the Flashback algorithms. While some findings were inconclusive, we discussed potential reasons and directions for deeper investigations.

While out of the scope of the original project, it will be interesting to see how the federated learning method performs on larger datasets that contain information from multiple locations, such as a countrywide dataset or a worldwide dataset. We suspect that in such cases a more thorough experimental setup and a more robust aggregation strategy would be necessary in order to account for the inherent data and device heterogeneity caused by the large geographical distribution of the data and the different time zones. Additionally, it would be interesting to explore federated personalisation techniques in order to fine-tune the global model on each client using their own local data. This is typically done for a small number of epochs with the aim of further improving individual performance while maintaining privacy. Flashback contains hints of personalisation with the use of user-specific embedding layers, but it remains an open question whether our findings extend to more sophisticated and state-of-the-art personalised recommendation architectures.

References

[1] Md Ashraful Islam, Mir Mahathir Mohammad, Sarkar Snigdha Sarathi Das, and Mohammed Eunus Ali. A survey on deep learning based Point-of-Interest (POI) recommendations. *Neurocomputing*, 472:306–325, 2022.

[2] Vasileios Perifanis, George Drosatos, Giorgos Stamatelatos, and Pavlos S. Efraimidis. FedPOIRec: Privacy Preserving Federated POI Recommendation with Social Influence, December 2021. arXiv:2112.11134 [cs].

[3] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. Flower: A friendly federated learning research framework, 2022.

[4] Marta C Gonzalez, Cesar A Hidalgo, and Albert-Laszlo Barabasi. Understanding individual human mobility patterns. *nature*, 453(7196):779–782, 2008.

[5] Dingqi Yang, Benjamin Fankhauser, Paolo Rosso, and Philippe Cudre-Mauroux. Location prediction over sparse user mobility traces using RNNs: Flashback in hidden states! In *Proceedings of the twenty-ninth international joint conference on artificial intelligence*, pages 2184–2190, 2020.

[6] Zhu Sun, Yu Lei, Lu Zhang, Chen Li, Yew-Soon Ong, and Jie Zhang. A Multi-Channel Next POI Recommendation Framework with Multi-Granularity Check-in Signals, September 2022. arXiv:2209.00472 [cs].

[7] Zehua Sun, Yonghui Xu, Yong Liu, Wei He, Lanju Kong, Fangzhao Wu, Yali Jiang, and Lizhen Cui. A survey on federated recommendation systems. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

[8] Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. Predicting the next location: A recurrent model with spatial and temporal contexts. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

[11] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE transactions on information forensics and security*, 15:3454–3469, 2020.

[12] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*, 2016.

[13] Yeting Guo, Fang Liu, Zhiping Cai, Hui Zeng, Li Chen, Tongqing Zhou, and Nong Xiao. PREFER: Point-of-interest recommendation with efficiency and privacy-preservation via federated edge learning. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5(1):1–25, 2021.

[14] Li-e Wang, Yihui Wang, Yan Bai, Peng Liu, and Xianxian Li. POI recommendation with federated learning and privacy preserving in cross domain recommendation. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE, 2021.

[15] Vasileios Perifanis, George Drosatos, Giorgos Stamatelatos, and Pavlos S Efraimidis. FedPOIRec: Privacy-preserving federated POI recommendation with social influence. *Information Sciences*, 623:767–790, 2023.

[16] Dingqi Yang, Daqing Zhang, and Bingqing Qu. Participatory Cultural Mapping Based on Collective Behavior Data in Location-Based Social Networks. *ACM Transactions on Intelligent Systems and Technology*, 7(3):30:1–30:23, January 2016.

[17] Yang Li, Tong Chen, Yadan Luo, Hongzhi Yin, and Zi Huang. Discovering Collaborative Signals for Next POI Recommendation with Iterative Seq2Graph Augmentation, April 2022. arXiv:2106.15814 [cs].

A Graph presentation: Matplotlib open issue

Following the project guidelines, we tried to make our plots black-and-white friendly by using the hatching feature of Python's `matplotlib` library. However, this conflicts with the requirement for plots to be saved in vector (PDF or SVG) format: the hatches only appear in rasterised outputs. This is an open issue and related issues can be tracked via the following links:

- `https://github.com/matplotlib/matplotlib/issues/12367`
- `https://github.com/matplotlib/matplotlib/issues/16052`
- `https://github.com/matplotlib/matplotlib/issues/16883`

All potential solutions (such as setting alpha to a higher value) were tried but to no avail. It is therefore strongly recommended that this report be viewed in colour. *For the benefit of future L361 students, perhaps a code workaround could be provided, or the requirements for black-and-white friendliness be relaxed.*

B Foursquare Dataset

B.1 Raw dataset

Table A4 characterises the datasets that we use as a starting point before pre-processing. Note that the raw dataset is not directly used by either MCMG or Flashback; for what is actually used, refer to Section B.2.

Table A4: Summary of the datasets used in the experiments.

Dataset	Check-ins	Users	Locations
`CAL`	9,317	130	604
`PHO`	35,337	767	1,950

The raw datasets for each city are obtained as CSV files directly from the original MCMG repository at `https://github.com/2022MCMG/MCMG/tree/main/dataset`.

B.2 After pre-processing

We investigate three levels of quantity skew in hypothesis H2: *Very heterogeneous*, *Slightly heterogeneous* and *Approximately homogeneous*. This section outlines dataset characteristics of the three corresponding datasets. The way these are generated are described

Table A5: MCMG: Very heterogeneous.

Dataset	Check-ins	Users	Locations
`CAL`	2,226	81	580
`PHO`	6,903	377	1,847

Table A6: MCMG: Slightly heterogeneous.

Dataset	Check-ins	Users	Locations
CAL	1,071	8	178
PHO	2,845	20	536

Table A7: MCMG: Approximately homogeneous.

Dataset	Check-ins	Users	Locations
CAL	7	7	7
PHO	47	47	42

Table A8: Flashback: Very heterogeneous.

Dataset	Check-ins	Users	Locations
CAL	5,742	27	472
PHO	17,723	92	1,479

Table A9: Flashback: Slightly heterogeneous.

Dataset	Check-ins	Users	Locations
CAL	4,713	25	444
PHO	14,010	86	1,360

Table A10: Flashback: Approximately homogeneous.

Dataset	Check-ins	Users	Locations
CAL	822	5	138
PHO	1,968	14	406

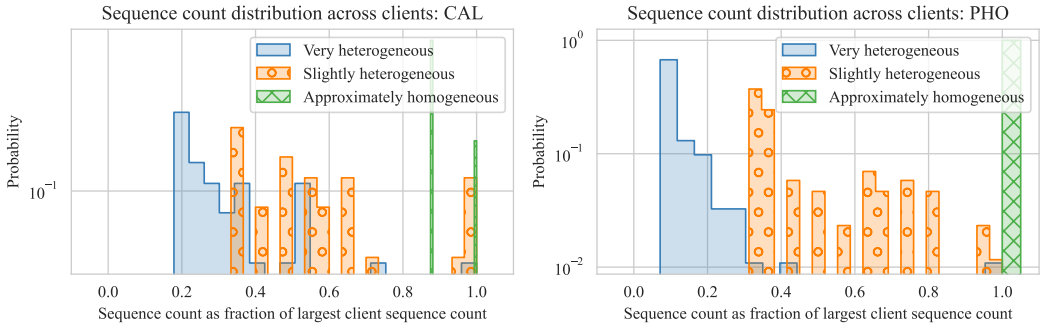

Figure 2: Dataset size distribution of the three reduced datasets used to test H2 using Flashback. The original dataset (blue) shows the greatest quantity skew heterogeneity (spread on the horizontal axis), followed by the slightly heterogeneous (orange) case, then the approximately homogeneous (green case).

C Model Hyperparameters

Tables A11 and A12 outline the hyperparameters used during training.

Considering the way the original Flashback implementation was coded, the intent to reuse as much of the original code as possible, and the requirement to adapt it to run on the Flower framework:
*The 100 epochs were split into 20 rounds of 5 epochs each for the federated case within the Flower framework.
† The total number of locations and users ($|P|$ and $|U|$ respectively) in the city are dataset specific and must be known in advance.
‡ There is a runtime restriction that forces `batch_size` to be at most as large as `user_count`. In our implementation, we set $f(|U|)$ to be the largest power of two at most $|U|$.

Table A11: Summary of the hyperparameters used for training MCMG models on the different datasets.

Hyperparameter	CAL	PHO
batch size	512	512
epochs	100	100
l2	0.0001	0.0001
patience	10	10
heads	1	1
blocks	1	1
GCN drop out	0.5	0.5
SA drop out	0.5	0.5
GCN drop out	0.5	0.5
tune epochs	30	30

Table A12: Summary of the hyperparameters used for training Flashback models on the different datasets.

Hyperparameter	CAL	PHO
batch size	$f(\|U\|)^{\ddagger}$	$f(\|U\|)^{\ddagger}$
epochs	100*	100*
loc_count	$\|P\|^{\dagger}$	$\|P\|^{\dagger}$
user_count	$\|U\|^{\dagger}$	$\|U\|^{\dagger}$
hidden_dim	10	10
lambda_t	0.1	0.1
lambda_s	100	100

D Experiment commands

All experiments can be performed within the confines of the Flower framework. Here are the procedures to reproduce all findings in this report:

D.1 MCMG

1. Clone the repository derived from the Flower template[5]
2. Run `./setup.sh`
3. Run `poetry run python -m project.task.mcmg.dataset_preparation` to download and pre-process the datasets
4. Run `poetry run python -m project.main --config-name=mcmg` to run the MCMG algorithm for H1.
5. Modify `mcmg.yaml` to create additional scenarios.

D.2 Flashback

1. Clone the repository derived from the Flower template[6]
2. Run `./setup.sh`
3. Run `poetry run python -m project.task.flashback.dataset_preparation` to download and pre-process the datasets
4. Run `poetry run python -m project.task.flashback.generate_experiment_yamls` to generate configuration YAML files used for all experiment configurations
5. (a) Run `poetry run python -m project.task.flashback +task/flashback='glob(*)'` to run the Flashback algorithm for H1, H2 and H3.
   These were executed on a CPU and took roughly 3+ hours to complete.
   (b) Run `poetry run python -m project.task.flashback +task/flashback-nonfederated='glob(*)'` to run the Flashback algorithm for H1.
   This was executed on a CPU and took roughly 6+ hours to complete.
6. Step through the notebook in `plotting/data_analysis_flashback.ipynb` to obtain the results and plots.

[5] `https://github.com/jackykwe/l361-project-next-poi-prediction`
[6] `https://github.com/jackykwe/l361-project-next-poi-prediction`

E Reproducibility Statement

The original MCMG[7] and Flashback[8] implementations are open-source and available on Github. Their adaptations into the Flower framework are also open-source and available on Github.[9] Code used to run single-client MCMG experiments are also open-source and available on Github.[10] The raw datasets for each city are obtained as CSV files directly from the original MCMG repository[11]; we have copies of them should the MCMG repository go offline, and may be provided on request. Data processing steps (of the same dataset, separately for both MCMG and Flashback) are outlined in Sections 4.2.1–4.2.2, and low-level details may be inspected by inspecting the source code of our implementation.[12] Steps to reproduce experiments are outlined in Appendix D. Proper library versioning of packages and Python virtual environments is handled using the `poetry` Python library, as provisioned by default via the provided repository template.[13]. All sources of randomness (`random`, `np.random`, `torch.random`) are seeded via the configuration files at `fed.seed`, as again provisioned by default via the provided repository template, whose documentation describes this in further detail.[14] The exact seeds we used are shown in Table A13; for each model, we stuck with the same three seeds for all experiments.

Table A13: Random generator seeds used to produce results in this report.

Model	Seed 1	Seed 2	Seed 3
MCMG	1337	1338	1339
Flashback	42	361	1337

F Interpretation of Flower's History File

This section is primarily for our future reference. It may additionally also serve as useful documentation for new users of the Flower framework, whose documentation is currently incomplete.

An overview of the federated learning stages can be found at Flower's documentation for strategies. In particular, this is how we interpret the different parts of the `history.json` file produced after each experiment. These were verified by tracing function calls and liberal use of print statements.

- `losses_distributed`: Starts with round 1. (Losses are returned via both the `train_test.train()` and `train_test.test()` method.)

  These are **distributed test losses** obtained by training on each client's local dataset, during the *1: Federated Training* stage. Values are weighted averaged across all evaluation clients, weighted by the train dataloaders' sizes.

- `losses_centralized`: Starts with round 0. (Losses are returned via both the `train_test.train()` and `train_test.test()` method.)

  These are **centralised test losses** obtained by testing on the server's federated dataset, during the *2: Centralised Evaluation* stage.

- `metrics_distributed_fit`: Starts with round 1. Includes only `train_loss`. (Losses are returned via both the `train_test.train()` and `train_test.test()` method.)

  These are **distributed train losses** obtained by training on each client's local dataset, during the *1: Federated Training* stage. Values are weighted averaged across all evaluation clients, weighted by the train dataloaders' sizes.

[7] `https://github.com/2022MCMG/MCMG`

[8] `https://github.com/eXascaleInfolab/Flashback_code`

[9] `https://github.com/jackykwe/l361-project-next-poi-prediction`

[10] `https://github.com/DobromirM/MCMG`

[11] `https://github.com/2022MCMG/MCMG/tree/main/dataset`

[12] See footnote 9

[13] `https://github.com/camlsys/fl-project-template`

[14] `https://github.com/jackykwe/l361-project-next-poi-prediction?tab=readme-ov-file#reproducibility`

- `metrics_distributed`: Starts with round 1. Includes all evaluation metrics used by our models. (These are only returned via the `train_test.test()` method.)

  These are **distributed test metrics** obtained by testing on each client's local dataset, during the *3: Federated Evaluation* stage. Values are weighted averaged across all evaluation clients, weighted by the test dataloaders' sizes.

- `metrics_centralized`: Starts with round 0. Includes all evaluation metrics used by our models. (These are only returned via the `train_test.test()` method.)

  These are **centralised test metrics** obtained by testing on the server's federated dataset, during the *2: Centralised Evaluation* stage.