Spider: Deep Learning-driven Sparse Mobile Traffic Measurement Collection and Reconstruction

Yini Fang
School of Informatics
University of Edinburgh, UK
yini.fang@ed.ac.uk

Alec F. Diallo
School of Informatics
University of Edinburgh, UK
alec.frenn@ed.ac.uk

Chaoyun Zhang
Lightspeed & Quantum Studios
Tencent, China
vyokkyzhang@tencent.com

Paul Patras
School of Informatics
University of Edinburgh, UK
paul.patras@ed.ac.uk

***Abstract*—Data-driven mobile network management hinges on accurate traffic measurements, which routinely require expensive specialized equipment and substantial local storage capabilities, and bear high data transfer overheads. To overcome these challenges, in this paper we propose Spider, a deep-learning-driven mobile traffic measurement collection and reconstruction framework, which reduces the cost of data collection while retaining state-of-the-art accuracy in inferring mobile traffic consumption with fine geographic granularity. Spider harnesses Reinforcement Learning and tackles large action spaces to train a policy network that selectively samples a minimal number of cells where data should be collected. We further introduce a fast and accurate neural model that extracts spatiotemporal correlations from historical data to reconstruct network-wide traffic consumption based on sparse measurements. Experiments we conduct with a real-world mobile traffic dataset demonstrate that Spider samples 48% fewer cells as compared to several benchmarks considered, and yields up to 67% lower reconstruction errors than state-of-the-art interpolation methods. Moreover, our framework can adapt to previously unseen traffic patterns.**

I. Introduction

Mobile traffic consumption continues to grow sharply and, as 5G is rolled out, the volume of data traffic per month globally is expected to hit a staggering 226 exabytes milestone by 2026 [1]. In this context, network visibility is becoming critical to the management of resources and to assuring network performance. Data-driven network management hinges on accurate traffic measurements [2] collected by dedicated probes that are deployed, e.g., at Packet Gateways (PGWs) [3]. Yet processing vast amounts of data in a scalable and timely fashion is ever more challenging, as it involves substantial local storage, heavy overhead in transferring detailed logs to central locations for analysis, data filtering by scope (e.g., cell ID, session start/end time, traffic volume/type, etc.) to serve specific use cases [4], and it relies on high-performance computing platforms to extract essential insights. Therefore mobile operators urgently need more cost-effective alternative solutions for traffic monitoring and analysis.

Adaptive sampling and compressed sensing have been employed to simplify the network traffic characterization task, whereby only a subset of data collection points are activated and/or monitoring is performed during sparse intervals. Complete network traffic snapshots are subsequently reconstructed via interpolation [5]–[7]. Although such methods adjust the sampling frequency based on previously measured network activity, sampling locations are routinely selected at random, which *(i)* overlooks important spatiotemporal correlations specific to mobile traffic, leading to inaccurate reconstruction; and *(ii)* such myopic view leaves room for sampling overhead reduction, without sacrificing interpolation fidelity.

More recently, neural network models have been proposed to predict data traffic demand in large coverage areas when only incomplete measurements are available [8], or to perform mobile traffic super-resolution, i.e., infer traffic consumption with fine geographic granularity, based only on coarse aggregate measurements [9]. Although these approaches obtain good-quality interpolations by harnessing the exceptional abstract feature extraction abilities of deep learning, *they are built on the premise that the measurement collection points are fixed*. In practice, mobile operators need intelligent strategies to instrument dynamic measurement collection campaigns using virtual probe modules that can be instantiated as needed.

In this paper, we propose Spider, a deep learning-driven mobile traffic measurement collection and reconstruction framework for infrastructure-level data. Spider relies on a dedicated neural network that we train to selectively sample small subsets of target mobile coverage areas. It then employs a purpose-built mobile traffic reconstruction neural model, which exploits spatiotemporal correlations in historical data, to infer mobile traffic consumption across the entire deployment. Our framework reduces dramatically the cost of data collection while retaining high traffic consumption inference accuracy. In summary, we make the following **key contributions**:

(1) We introduce a policy network that takes as input sparse historical traffic snapshots and outputs cell selection matrices indicating at which locations to activate measurement collection, so as to minimize overhead while acquiring enough data to ensure high-quality traffic consumption interpolations at all non-sampled cells. We take a Deep Reinforcement Learning (DRL) approach to produce examples for training this policy network. Given the large action space, our DRL agent learns in a tractable manner by sampling small subsets of the action space based on the most likely action and its nearest neighbors. The highest valued action from such subsets is then selected, circumventing the need to evaluate the entire action space.

(2) We propose MTRNet, a deep neural network specifically tailored to mobile traffic reconstruction from sparse traffic

measurements. MTRNet outperforms existing methods in terms of accuracy (up to 67% lower MAE) and reconstruction speed (up to 835 runtime reduction).

(3) We evaluate our Spider framework using a real-world mobile traffic dataset collected by a major European operator, showing that our solution learns to effectively reconstruct complete traffic matrices from sparse samples, even when applied to previously unseen traffic patterns such as those observed during holidays. On average, Spider accurately reconstructs city-scale traffic snapshots with up to 48% fewer samples than a range of benchmarks considered.

II. Problem Formulation

Consider a mobile network coverage area that is geographically partitioned into a grid with $X \quad Y$ squares (cells), where a square denotes an atomic region for mobile traffic collection. We denote by F_t the traffic consumption snapshot across all cells, observed over an interval $[t \quad ; t]$, i.e.,

$$F_t = \begin{bmatrix} d^t_{1;1} & d^t_{1;2} & \cdots & d^t_{1;Y} \\ d^t_{2;1} & d^t_{2;2} & \cdots & d^t_{2;Y} \\ \vdots & \vdots & \ddots & \vdots \\ d^t_{Y;1} & d^t_{Y;2} & \cdots & d^t_{X;Y} \end{bmatrix};$$

where $d^t_{i;j}$ is the volume of traffic at cell $(i; j)$, and is the time granularity configurable by a network administrator, with which traffic measurement equipment (probes) compute summaries of the volume of traffic observed at different locations. A mobile network operator will need a sampling strategy that gives a binary selection matrix B_t indicating which cells are to be selected for measurement collection at time t, i.e., $b^t_{i;j} = 1$, if cell $(i; j)$ is selected; $b^t_{i;j} = 0$, otherwise; and subsequently use the measurements collected to infer the traffic consumption across the entire deployment. A selection matrix produces a sparse measurement matrix M_t of a network traffic snapshot F_t, i.e.

$$M_t = F_t \quad B_t;$$

where is the Hadamard product. Let $\mathsf{M}_t = [M_{t \quad T}; \ldots; M_t]$ denote the sparse measurement matrices collected over T recent timestamps, and $\hat{F}_t$ the reconstruction of F_t given M_t.

The problem we seek to solve is thus twofold: *(i)* finding a policy $g(\,)$ that outputs cell selection matrices $\hat{B}_t = g(\mathsf{M}_t)$, which contain minimum numbers of elements $b^t_{i;j}$ set to 1, so that the error of reconstructing F_t using those measurements collected is below a predefined threshold , by using *(ii)* a data interpolation algorithm $f(\,)$. Formally,

$$\hat{B}_t := \arg\min_{B_t} \sum_{i;j} b^t_{i;j}; \tag{1}$$

$$\text{s.t. } \mathrm{MAE}(\hat{F}_t; F_t) < \ ; \tag{2}$$

$$\hat{F}_t = f(\mathsf{M}_t): \tag{3}$$

In the above MAE is the Mean Absolute Error, i.e.,

$$\mathrm{MAE}(\hat{F}_t; F_t) = \frac{1}{X \quad Y} \sum_{i;j} j\hat{d}^t_{i;j} \quad d^t_{i;j}j: \tag{4}$$

We solve this problem using a set of purpose-built deep neural network models as we explain next.

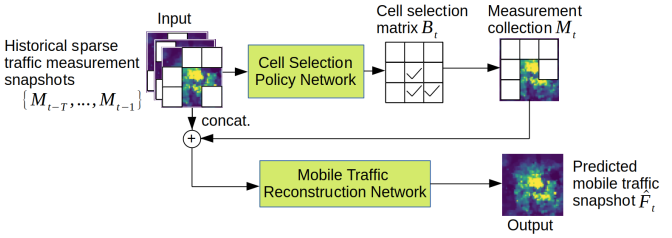

Fig. 1: Proposed Spider framework: sparse mobile traffic snapshots used by a policy network to select optimal cells where to collect measurements; a dedicated reconstruction neural model outputs completed network traffic snapshot.

III. Mobile Traffic Measurement Collection and Reconstruction Framework

To solve the problem defined by (1)–(3), we propose Spider, an original mobile network traffic measurement collection and reconstruction framework that *(i)* predicts a small number of optimal locations where traffic should be sampled, and *(ii)* learns to infer the volume of traffic at locations not selected, so as to reconstruct complete snapshots of the traffic consumption across an entire network deployment. We give a diagramatic view of our approach in Fig. 1. We design original neural models that harness the unique characteristics of mobile traffic to solve both of these tasks.

A. RL for Cell Selection with Large Action Spaces

To collect measurements with minimum sampling overhead, we first leverage Reinforcement Learning (RL) and train an agent that learns the likelihood of selecting a cell where to sample traffic, based on past experience. We subsequently use this agent to train a policy network that *directly outputs optimal cell selection matrices*, given prior observations, and thus is suitable for real-time decision making.

We start by regarding cell selection as an episodic task, which can be modelled as a Markov Decision Process (MDP), $M := (S; A; P; r; \ )$, where

S is the set of states s_i, in our case a state representing a collection of sparse measurement matrices of the past T network traffic snapshots, i.e., $s_i = \mathsf{M}^i_t = [M_{t \quad T}; \ldots; M^{i \quad 1}_t]$, with $M^{i \quad 1}_t$ denoting the sparse measurement matrix at iteration $i \quad 1$ of an episode t;

A is the set of possible actions, where an action a_i corresponds to sampling one of all the previously unselected cells at timestamp t, from which the agent can choose;

$P(s; a; s')$ is the probability that action a in state s will lead to next state s';

$r(s; a; s')$ is the reward the agent receives as a consequence of choosing action a when in state s; we work with $r(s; a; s') = \quad \mathrm{MAE}(f(s'); F_t)$ to incentivize the agent to take actions that reduce the reconstruction error; while a range of methods $f(\,)$ can be employed for reconstruction, including compressive sensing or K-Nearest Neighbour-based interpolation, our Spider framework evaluates how good an action is using a pre-

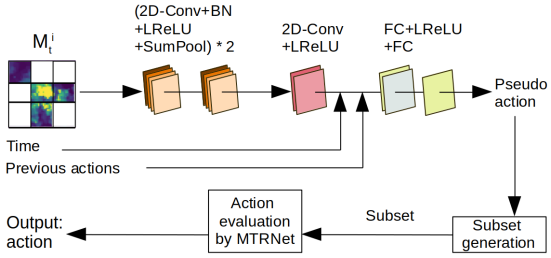

Fig. 2: Structure of the DRL agent. The pseudo action given by the neural network is used to generate a set of potential candidates that, when evaluated by MTRNet, allows the agent to select the action that minimizes the reconstruction error.

trained Mobile Traffic Reconstruction neural Network (MTRNet), which we detail in Sec. III-B.

2 $[0;1]$ is the discount rate – an agent's objective is to maximize a cumulative reward it receives (expected return), i.e. the sum of discounted rewards $r^{k\ 1}$ over k iterations; we work with $= 0$, meaning that at each iteration we seek to maximize the immediate return.

We consider a cell selection episode t to be completed at an iteration I when the inference error is less than a predefined reconstruction quality threshold.

With traditional DRL, finding a policy that maps from states to probabilities of choosing an action in a large action space such as city-scale mobile network deployments comprising thousands of cells would require massive amounts of memory. Further, exploring the action space exhaustively, until sufficient cells are selected to yield usable reconstructed traffic snapshots, would involve impractically large runtimes. To address these issues, we design a DRL agent based on a neural network architecture to model the policy function, which we train using an efficient algorithm that only evaluates a subset of potentially good actions from the whole action space, among which the best action is selected following evaluation. As such the agent will first output a pseudo action $\hat{a}$, which is used to generate a subset of k potential actions by finding the K-Nearest Neighbors of $\hat{a}$ in A based on $`_2$ distance, i.e.,

$$A_k^0 = knn(\hat{a}) = \underset{a2A}{\overset{k}{\arg\min}} jja \quad \hat{a}jj_2;$$

because the spatial correlation between two cells is related to their distance. To explore previously unseen actions, the generated subset A_k^0 is expanded with randomly selected actions $rand\ (A_k^0)$ not already in A_k^0, where is given by an exponentially decaying function, i.e.

$$A^0 := A_k^0 \quad + rand\ (A_k^0 \quad);$$

where $= bk(0{:}1 + 0{:}9{=}e^x)e$ and x is the number of training episodes completed. This approach reduces both memory requirements and execution time by constraining the search space, and favors exploration at the beginning, when 100% of the actions are chosen randomly.

We illustrate the DRL agent's structure in Fig. 2 and present the training process in Algorithm 1. The agent takes as input the sparse measurements snapshot $M_t^{i\ 1}$ built up to the current

Algorithm 1: Training the RL agent

1 Input: Pre-trained MTRNet $f()$, environment E, the number of epochs n_e, the number of snapshots in training data n_s, the number of historical snapshots used T;
2 Initialize the neural network $p()$;
3 **for** *epoch* $\leftarrow 1$ **to** n_e **do**
4 Initialize E;
5 **for** $t \leftarrow T + 1$ **to** n_s **do**
6 **while** *episode not finished* **do**
7 Get current state $s_i = [M_{t-T}, ..., M_t^{i-1}]$, previously selected actions A, and the ground truth snapshot F_t from E for the current iteration i;
8 Generate subset A' from A and $p(M_t^{i-1}, t, \mathsf{A})$;
9 $e = \{\}$;
10 **for** a *in* A' **do**
11 Get s_j by applying a to s_i;
12 $e_j = MAE(f(s_j), F_t)$;
13 Push e_j to e;
14 **end**
15 $a' = \underset{a \in \mathcal{A}'}{\arg\min}(e)$;
16 Apply action a' to E;
17 $Loss = \|p(M_t^{i-1}, t, \mathsf{A}) - a')\|_2^2$;
18 Use $Loss$ to update $p()$ by gradient descent;
19 **end**
20 **end**
21 **end**

iteration i, the list of cells selected so far, and the time t. From the sparse measurements matrix, the neural network first extracts feature maps using 2D convolution layers, with a sum pooling layer applied between them to summarize these feature maps without diluting the active features. This enables faster learning of the traffic patterns observed within a geographical window. The resulting features are then fed to a 2D convolution layer to reduce the size of the aggregate feature map, such that each feature's magnitude is of the same scale as the time and previous actions. Two fully connected layers then process the aggregate feature map, along with the time and the previous actions, to predict the next action. Based on this value, the agent generates a set of potential candidates to be evaluated by MTRNet (Algorithm 1, lines 10–14), and the candidate yielding the smallest reconstruction error is chosen by the agent as next action (line 15). Finally, the neural network learns to improve its predictions by updating its weights based on gradients of the Mean Squared Error loss between the predicted (pseudo) action and the chosen candidate (lines 17–18). For any given snapshot, this process is repeated until the reconstruction error is less than a predefined reconstruction quality threshold.

In choosing this threshold we seek a trade-off between achieving low reconstruction errors and selecting a small number of cells. To find an appropriate quality threshold we examine the MAE reduction as a function of (random) cell sampling rates and observe that MAE decreases only marginally beyond 35% sampling rates. Hence we work with this value as the quality threshold.

When predicting the pseudo action, the agent only considers the current snapshot instead of the current state. This

is because in choosing actions one by one, only one value in a matrix changes from zero to the measurement between two consecutive states, which would make the collected value less important in the input, wrt. historical data. Although the previous experience is not seen in the pseudo action prediction, MTRNet evaluates an action given the full state as the input, whereby temporal correlations are captured.

B. Traffic Reconstruction from Sparse Measurements

We perform traffic measurement reconstruction using a supervised learning approach. The proposed Mobile Traffic Reconstruction neural Network (MTRNet) takes as input the sparse measurement matrices for the current (t-th) and previous T timestamps, i.e., $[M_{t-T+1}, \ldots, M_t]$, and outputs the traffic consumption of the full map $\hat{F}_t$ at the current timestamp.

The model architecture is shown in Fig. 3. We draw inspiration from ZipNet [9], a mobile traffic super-resolution technique that infers fine-grained traffic consumption from coarse measurements. The original ZipNet was modified in order to reduce the size of the model and improve its inference speed. Specifically, since in our setting the size of sparse measurement matrices is the same as the size of the output, we discard upscaling blocks. Further ablation study allows us to reduce the number of layers of the model (depth) while retaining high accuracy. The resulting MTRNet comprises three key components:

Correlations capturing. A 3D convolutional layer is used to capture spatiotemporal correlations between recent snapshots. The activation layer is a Leaky-ReLU (LReLU) that improves the model's non-linearity and its robustness to network initialization. LReLU is defined by:

$$\text{LReLU}(x) = \begin{cases} x, & x \geqslant 0 \\ \alpha x, & x < 0 \end{cases},$$

where α is a configurable slope value. A SumPooling layer is applied after LReLU to reduce the size of the feature maps and speed up the training.

Feature extraction. Several 2D Convolutional layers followed by a LReLU extract high level abstract features from the geographical configuration of the measured traffic, thereby enhancing the representability of the model. Staggered skip connections link every two blocks, and a global skip connection links the input and output of this component. These skip connections hierarchically preserve different features extracted from previous layers, enabling feature reusability and stabilizing training and convergence.

Feature summarization. To maximize the probability of capturing all relevant combinations of abstract features extracted by the previous component, the first two layers of the feature summarization block gradually increase the number of convolution filters (or kernels). Subsequently, the resulting feature maps are merged by a third and last convolution layer to provide a faithful reconstruction of the complete traffic snapshot, which represents the final output of our proposed MTRNet.

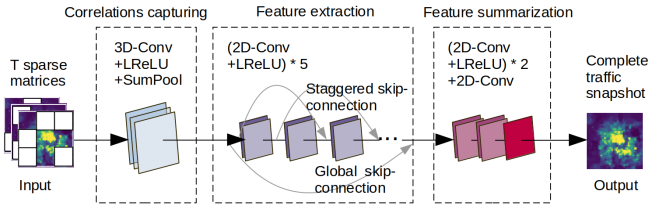

Fig. 3: MTRNet structure comprising correlations capturing, feature extraction & summarization functionality. Traffic snapshots reconstructed based on historical sparse measurements.

C. Policy Network

Since RL agents typically evaluate one action at a time, they are often impractical to deploy in settings where decisions have to be made within short deadlines. Therefore, we design a policy network that *predicts all optimal cell locations to be sampled, $B_t = g(M_t)$, at once*. This policy network (g) takes historical sparse measurement snapshots as input and learns to predict binary selection matrices generated by the agent introduced in Sec. III-A at the end of each episode.

This neural network effectively solves a multi-label classification task where positive labels represent cells to be selected and negative labels indicate non-selected squares. The model architecture is similar to that of MTRNet, though here a final Sigmoid layer transforms the outputs into probability scores and we assign positive labels to those selection matrix elements where the probability is above 0.5, and negative to all others. As the cell selection frequencies are likely to be skewed, the probabilities are normalized to the range $[0, 1]$ before the binarization process.

We use a Binary Cross-entropy (BCE) loss function to train this neural model, which is defined by $BCE = \frac{1}{N}\sum_{i=0}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$, where y_i is the value in the binary selection matrix, and $\hat{y}_i$ is the output probability score. N denotes the total number of elements (cells).

IV. Evaluation

In this section we evaluate the performance of our Spider framework in terms of the number of cells sampled for mobile traffic visibility, and measurement reconstruction error.

We use a compute cluster comprising 20+ nodes, each equipped with 1-2 NVIDIA TITAN X GPUs (2280 cores) to train the neural models. We implement Spider in Python using the PyTorch library, and train on 2 GPU nodes in parallel for 2 epochs, spending 9.6 hours on each GPU. The number of historical snapshot T is 6. For all the models, we employ the Adam optimizer with learning rate $= 10^{-4}$ and use 128 as the batch size. For MTRNet, the number of previous actions is 20 and the number of 2D Convolutional layers is 5.

A. Dataset

For evaluation we adopt a real-world mobile traffic dataset collected by Telecom Italia in the city of Milan [10]. The city is divided into 100 × 100 squares of 0.055 km^2. Mobile data traffic measurements were collected on aggregate at these locations every 10 minutes between 01/10/2013 and

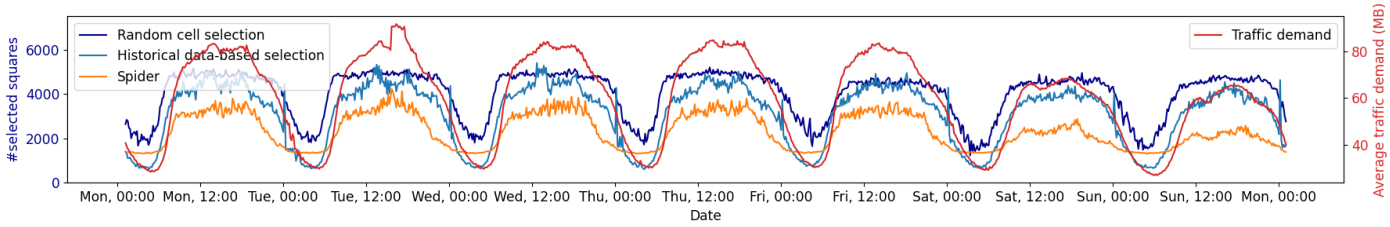

Fig. 4: Number of cells selected for measurement collection by Spider and benchmarks, over one week (data representative for 16–22 Dec 2013). Average volume of traffic also plotted to highlight Spider's ability to adapt to changing patterns.

TABLE I: Performance comparison between Spider and cell selection baselines, in terms of number of cells sampled and resulting NMAE following MTRNet-based reconstruction.

Interval	Random cell selection		Historical data-based selection		**Spider**	
	Count	NMAE	Count	NMAE	Count	NMAE
Peak	4,737	0.06	4,185	0.04	2,643	0.08
Off-peak	3,610	0.09	2,472	0.10	1,814	0.12
Weekdays	4,116	0.08	3,142	0.07	2,378	0.09
Weekend	3,838	0.08	3,033	0.07	2,237	0.09
Holiday	4,049	0.07	3,129	0.08	1,906	0.11
Overall	4,039	0.08	3,122	0.08	2,109	0.10

01/01/2014 (3 months). We use 40 days worth of data to train the models, and 20 days for testing. We normalize the traffic consumption and work with $x = \log(1 + x)/x$, where x is the traffic consumption and x is the mean of $\log(1 + x)$.

B. Measurement Collection

We first examine Spider's performance in terms of the average number of cells selected for measurement collection versus (1) a random selection strategy that chooses randomly which cells to sample, with their number being the average at any time of the day during different days of the week that yields reconstruction errors below the predefined threshold, as observed across the dataset used for training our agent; and (2) a strategy that chooses the most frequently selected cells based on past selection patterns. This frequency matrix is generated by averaging binary selection matrices obtained during training for the same time of the day and day of the week. To put things into perspective, we also compute the Normalised Mean Absolute Error (NMAE) obtained with our MTRNet when each of these strategies are employed.

We summarise the performances of Spider against the baselines considered in Table I, comparing NMAE and sampled cell count (rounded to the nearest integer) at peak and off-peak hours during a day, respectively during weekdays, weekends, and public holidays. Overall, **Spider samples 48.0% fewer cell than the random selection strategy and 32.4% fewer than the selection approach based on historical data**, at a negligible cost in terms of additional reconstruction error introduced relative to the average volume of traffic (NMAE).

We illustrate Spider's behavior and that of the benchmarks considered over an entire week (Monday to Sunday) using traffic from the testing set (red line) in Fig. 4. Observe that Spider captures accurately the changes in traffic demand and samples cells for measurement collection accordingly.

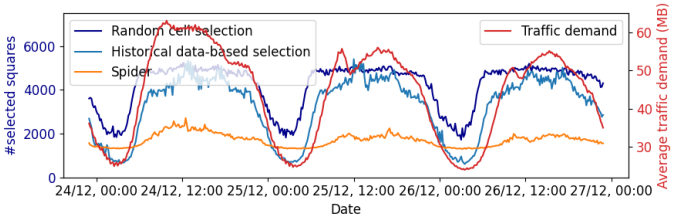

Fig. 5: Number of cells sampled by Spider and the benchmarks considered, during Christmas holidays.

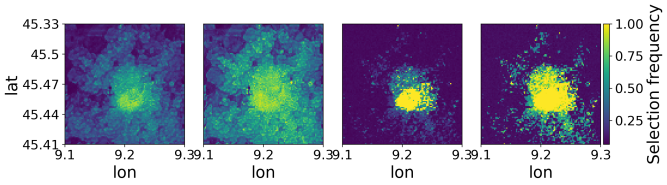

Fig. 6: Cell selection frequency. From left to right: historical data-based selection in off-peak, peak; Spider off-peak, peak.

While the historical data-based approach is able to distinguish between different times of the day and days of the week, it clearly over-samples, thus incurring higher measurement collection overhead. This is even more noticeable when cells are sampled randomly and with the only goal of meeting the quality threshold.

The superior performance of Spider is further emphasized in Fig. 5 where our framework is applied for measurement collection during the Christmas holidays, when traffic demand decreases below typical daily averages. Spider is able to adapt to previously unseen traffic patterns and distinguishes between holidays taking place on weekdays, and normal weekdays. Precisely, it selects 19.8% fewer cells on this occasion. In contrast, random and historical data-based selection approaches fail to adapt to such circumstances.

We delve deeper into which cells are selected by Spider, showing in Fig. 6 (right) their selection frequency at peak (7AM–7PM) and off-peak (7PM–7AM) times during weekdays, juxtaposed with the behavior of the historical data-based approach (left). Observe that Spider focuses more on the city center where the traffic demand is relatively larger, expanding sampling coverage at peak time.

Finally, we note that directly predicting a selection matrix by Spider's policy network takes 7 milliseconds on average, whereas predicting the next best action by the RL agent would have taken 2 seconds. Thus **our approach is 284 times faster and suitable for operational settings.**