

5 Appendix

The appendix is organized as follows:

1. Subsections 5.1 to 5.3 describe the preliminaries.
2. Subsection 5.4 explains the datasets, experimental parameters, and DNN architectures used in this work.
3. Subsections 5.5 to 5.9 give proofs for all theorems.

5.1 Why A Potential Game?

A potential function in a game is defined in Chapter 8 of [34]. It is a function indicating the incentives of all players (in our case robots), and any game with a potential is called a potential game. Typically, the goal of a player is to maximize its incentive expressed by the potential function. In our case, minimizing the loss function $\mathcal{L}(\rho_{\mathcal{D}_c^{r+1}}, \rho_{\mathcal{D}_{\text{target}}})$ in Eq. 3d is the common goal for all robots, so the potential function is the negative of the loss function, $-\mathcal{L}(\rho_{\mathcal{D}_c^{r+1}}, \rho_{\mathcal{D}_{\text{target}}})$. Note that the potential function $-\mathcal{L}(\rho_{\mathcal{D}_c^{r+1}}, \rho_{\mathcal{D}_{\text{target}}})$ is concave since the loss function $\mathcal{L}(\rho_{\mathcal{D}_c^{r+1}}, \rho_{\mathcal{D}_{\text{target}}})$ is convex.

5.2 Confusion Matrix

The confusion matrix of robot i at round r is defined as C_i^r , obtained by calculating the validation accuracy from a validation dataset. Its j -th row represents the probability vector of classifying a data-point of class j to different classes. If the confusion matrix is an identity matrix, it means that the classifier is perfect with 100% accuracy. The matrix is:

$$C_i^r = \begin{bmatrix} p_i^r(\hat{y}_1|y_1) & \dots & p_i^r(\hat{y}_{N_{\text{class}}}|y_1) \\ p_i^r(\hat{y}_1|y_2) & \dots & p_i^r(\hat{y}_{N_{\text{class}}}|y_2) \\ \dots & \dots & \dots \\ p_i^r(\hat{y}_1|y_{N_{\text{class}}}) & \dots & p_i^r(\hat{y}_{N_{\text{class}}}|y_{N_{\text{class}}}) \end{bmatrix}. \quad (4)$$

5.3 Calculating the correct conditional probabilities

As mentioned in Sec. 2, the robot's transmitted dataset a_i^r is calculated from the predicted class labels \hat{y}_j and not the true class labels y_j , which are not available on-robot. However, we can use predicted class probabilities $p_i^r(\hat{y}_j)$ to estimate true class probabilities $p_i^r(y_j)$ by: $p_i^r(y_j) = \sum_{k=1}^{N_{\text{class}}} p_i^r(\hat{y}_k) \cdot p_i^r(y_j|\hat{y}_k)$.

We can obtain the conditional probability $p_i^r(y_j|\hat{y}_j)$ by $p_i^r(y_j|\hat{y}_j) = \frac{p_i^r(\hat{y}_j|y_j) \cdot p_i^r(y_j)}{p_i^r(\hat{y}_j)}$ from the confusion matrix C_i^r and $p_i^r(\hat{y}_j)$ can be calculated from the model inference on robot i . Note that $p_i^r(y_j)$ can be *estimated* using a Bayesian Filter since we upload data at previous round $r-1$, which is assigned ground-truth labels.

5.4 Experiments

In the experiments, we simulated a system of multiple robots observing different image distributions $p_i^r(y)$ with the aim of sampling correct images to make the cloud dataset \mathcal{D}_c^r as close as possible to the uniform target dataset $\mathcal{D}_{\text{target}}$. First, an initial dataset \mathcal{D}_c^0 with random class distributions is selected to train the initial classification model $f(x; \theta_i^0)$. Next, the classification model is trained on the initial dataset \mathcal{D}_c^0 and its confusion matrix C_i^0 is calculated on the validation dataset. All robots have the same vision model in a simulation, thus the same confusion matrix. However, their incoming class distributions $p_i^r(y)$ are quite different, so each robot's feasible space H_i^r is different. In each round r , the robots label the images with their own classification model and solve the convex optimization problem to determine label allocations in the cache. At the end of each round r , sampled images are uploaded to the shared cloud, labeled by a human expert, and added to the cloud dataset with the correct labels. The cloud dataset statistics are updated and shared with all robots. When all sampling rounds are finished, the classification model is retrained on the final cloud dataset, which contains

the initial dataset and sampled images from all robots. All the DNNs are written in PyTorch, and the CVXPY package is used to solve the convex optimization problem.

In all experiments, we have divided our dataset into three non-overlapping parts: training, validation, and testing datasets. The training dataset is used to create the initial datasets and the images observed by robots. The validation and testing datasets are used to calculate the confusion matrix and the final accuracy, respectively.

We now explain the datasets, the simulation parameters, training/validation/testing splits, and DNN training hyperparameters used in the simulations.

5.4.1 MNIST Dataset

The MNIST dataset is a digit classification dataset consisting of 70,000 28×28 grayscale images with ten classes. The dataset consists of 60,000 training images and 10,000 testing images.

Simulation Parameters: For the MNIST dataset, we simulated $N_{\text{robot}} = 20$ robots for 7 rounds each observing 2000 images and sharing only $N_{\text{cache}} = 2$ images with the cloud. The initial dataset size is set to $N_{\mathcal{D}^0} = 200$ and at the end of the rounds a dataset of size $N_{\mathcal{D}^7} = 480$ is accumulated.

Training, Validation, and Testing Split: We divided the original training dataset into training and validation datasets of sizes 54,000 and 6000, respectively, and used the original test dataset of size 10,000. Thus 0.3% of the overall dataset size is used in the initial vision model. At the end of data sharing, we uploaded 0.8% of the full MNIST standard dataset to train the vision model. Training a model on this dataset yields a final accuracy of 93.07% on the full held-out test dataset. This value is close to 99.91% state-of-the-art accuracy for the full training dataset, which is very good given that our scheme uploads only a *fraction* of the data.

DNN and Training Hyperparameters: We now describe the vision model for the classification task. A DNN with four convolutional layers and two fully connected layers with ReLU activation layers is used as the classification model. Between the convolutional layers, dropout is applied with a rate of 0.3. In the convolutional layers, a kernel with a filter size of (3,3) and stride of 1 are used with a padding of 1. Finally, fully connected layers with sizes of (128,10) are used in consecutive fully connected layers. When the models are trained, a learning rate of 0.01 is used, and the batch size is set to 1000. We used the ADAM optimizer in training and used the exponential learning rate scheduler with a decay rate of 0.99. The DNN models are trained for 200 epochs. We only normalized the images before inputting them into the classification model.

5.4.2 CIFAR-10 Dataset

The CIFAR-10 dataset consists of 60000 $32 \times 32 \times 3$ RGB images with ten different classes. The original dataset is divided into training and testing datasets of sizes 50000 and 10000 respectively. This dataset contains 10 object classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

Simulation Parameters: For the CIFAR-10 dataset, we simulated $N_{\text{robot}} = 20$ robots for 5 rounds, each observing 5000 images and sharing only $N_{\text{cache}} = 200$ images with the cloud. The initial dataset size is set to $N_{\mathcal{D}^0} = 10000$ and at the end of the rounds a dataset of size $N_{\mathcal{D}^5} = 30000$ is accumulated.

Training, Validation, and Testing Split: We divided the original training dataset into training and validation datasets of sizes 45,000 and 5000, respectively, and used the original test dataset of size 10,000. Thus 20% of the overall dataset size is used in the initial vision model. At the end of data sharing, we uploaded 60% of the full CIFAR-10 dataset. Training a model on this dataset yields a final accuracy of 81.55% on the full held-out test dataset. This value is comparable to 91.25% state-of-the-art accuracy for the full training dataset, which is very good given that our scheme uploads only a *fraction* of the data.

DNN and Training Hyperparameters: We now describe the vision model for the classification task. A ResNet32 Model with 32 convolutional layers and skip connections is used as the classification model. We didn't use any pretrained weights for the vision model. When the models are trained,

a learning rate of 0.1 is used, and the batch size is set to 1000. We used the ADAM optimizer in training and used an exponential learning rate scheduler with a decay rate of 0.99. The DNN models are trained for 100 epochs. During training, we applied random cropping and random horizontal flips as data augmentation methods.

5.4.3 Adversarial-Weather Dataset

The `Adversarial Weather` dataset consists of thousands of $720 \times 1280 \times 3$ RGB image sequences collected in various weather conditions from moving vehicles. Most of the sequences are dynamic, while some are static recordings. The classes included in the dataset are rain, fog, snow, sleet, overcast, sunny, and cloudy. These weather conditions were recorded at various times of the day: morning, afternoon, sunset, and dusk. For the simulations, we have combined the time of day labels and the weather labels and created a total of 7 classes. Since the images are created from video sequences, we have subsampled the images once in every five frames to prevent having similar images. In the end, we created a dataset with 46025 images.

Simulation Parameters: For the `Adversarial Weather` dataset, we simulated $N_{\text{robot}} = 10$ robots for 5 rounds, each observing 2000 images and share only $N_{\text{cache}} = 20$ images with the cloud. The initial dataset size is set to $N_{\mathcal{D}^0} = 1000$ and at the end of the rounds a dataset of size $N_{\mathcal{D}^5} = 2000$ is accumulated.

Training, Validation, and Testing Split: We divided the dataset into training, validation, and test datasets of sizes 37279, 4143, and 4603, respectively. At the end of data sharing, training the model on the accumulated dataset yields a final accuracy of 94.27% on the full held-out test dataset.

DNN and Training Hyperparameters: A ResNet18 Model with 18 convolutional layers and skip connections is used as the classification model. We initialized the model weights with weights pre-trained on the ImageNet dataset and updated all layers. During training, a learning rate of 0.1 is used, and the batch size is set to 128. We used the ADAM optimizer during training and used the exponential learning rate scheduler with a decay rate of 0.99. The DNN models are trained for 50 epochs. During training, we first downsampled the images to a size of $256 \times 455 \times 3$, and applied random cropping and random horizontal flips as data augmentation methods.

5.4.4 DeepDrive Dataset

The `DeepDrive` dataset with 100,000 images is a driving video dataset from various cities in different weather conditions. The dataset consists of 70000 training, 10000 validation, and 20000 testing images. However, the testing images aren't publicly available. Therefore, we only used original training and validation datasets. We used the weather labels as the target of the classification model. The weather labels included in the datasets are: rainy, snowy, clear, overcast, partly cloudy, and foggy. We had to discard the foggy classes from the simulations because this class included only 181 images. Therefore, we trained the classification model on 5 classes.

Simulation Parameters: For the `DeepDrive` dataset, we simulated $N_{\text{robot}} = 20$ robots for 5 rounds, each observing 5000 images and sharing only $N_{\text{cache}} = 50$ images with the cloud. The initial dataset size is set to $N_{\mathcal{D}^0} = 8000$ and at the end of the rounds the dataset of size $N_{\mathcal{D}^5} = 13000$ is accumulated.

Training, Validation, and Testing Split: We divided the original training dataset into training and validation datasets of sizes 36968, 24646, respectively, and used the original validation dataset as the testing dataset of size 8830. Thus 11.43% of the overall dataset size is used in the initial vision model. At the end of data sharing, we uploaded 18.57% of the full `DeepDrive` dataset. Training a model on this dataset yields a final accuracy of 68.10% on the full held-out test dataset. This value is comparable to 81.57% state-of-the-art accuracy for the full training dataset, which is very good given that our scheme uploads only a very small fraction of the data. Moreover, our scheme beats the Greedy Benchmark by 12.4% as shown in Fig. 3.

DNN and Training Hyperparameters: A ResNet18 Model with 18 convolutional layers and skip connections is used as the classification model. We initialized the model weights with weights pre-

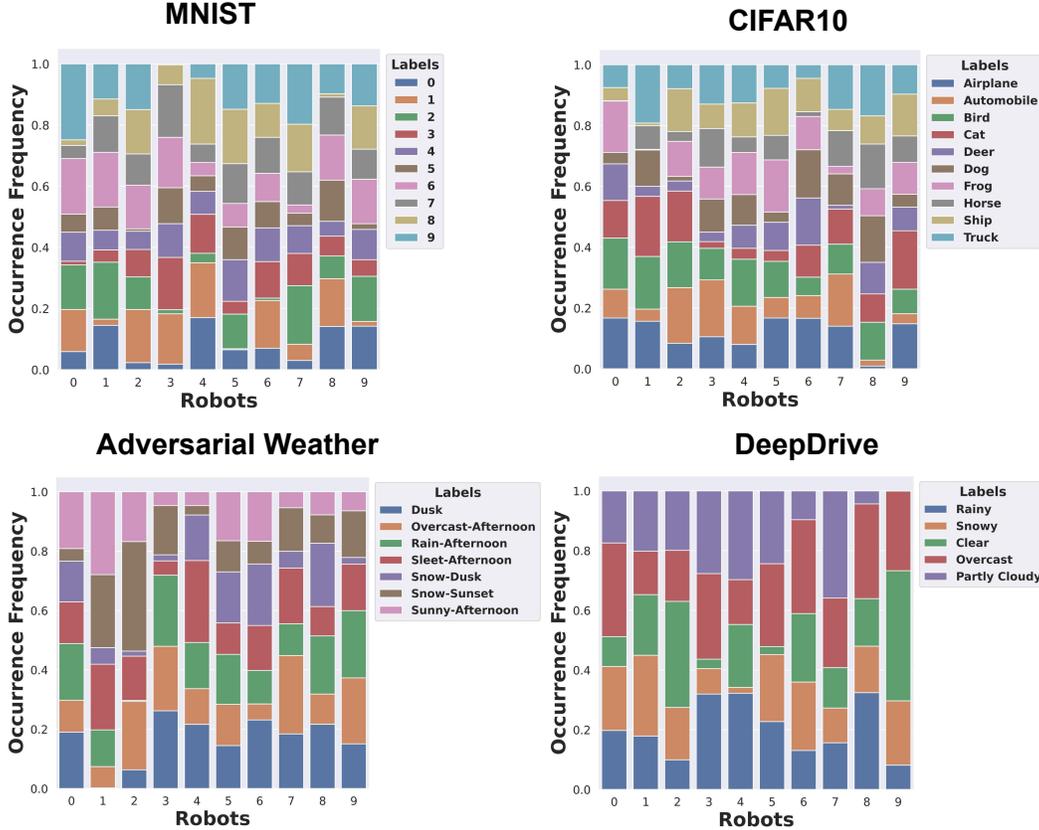


Figure 4: **The true class probabilities $p_i^r(y_j)$ of 10 randomly selected agents for experiments is non-uniform:** As expected for real-life robotics settings, different robots observe non-identical, skewed data distributions in our experiments. We randomly shuffled data for the synthetic datasets (MNIST/CIFAR). However, we plot *real-world* data distributions observed in the AV datasets. We randomly selected 10 agents for visual clarity.

trained on the ImageNet dataset and updated all layers. During training, a learning rate of 0.1 is used, and the batch size is set to 128. We used the ADAM optimizer in training and used the exponential learning rate scheduler with a decay rate of 0.99. The DNN models are trained for 50 epochs. During training, we first downsampled the images to a size of $256 \times 455 \times 3$, and applied random cropping and random horizontal flips as data augmentation methods.

5.4.5 Heterogenous Data Distributions for Robots

We now show that all experiments have heterogenous data distributions. This is a hallmark of real robotics settings that we observed from the real AV datasets. Fig. 4 illustrates that each agent (x-axis) has a markedly different data distribution of true class probabilities $p_i^r(y)$ than others (y-axis barplot). For the synthetic MNIST and CIFAR-10 datasets, we randomly shuffled data distributions across agents. However, we plot the *true* data distributions across AVs for the real-world autonomous driving datasets.

Fig. 5 shows the distribution of classes across different cities in the DeepDrive dataset is also heterogeneous. Clearly, a majority of rainy scenes are in New York and a majority of foggy scenes are in SF. Moreover, New York has the highest percentage of city street scenes.

Finally, the highly heterogenous distribution of classes can be seen in the map views from the DeepDrive dataset in Fig. 6. Clearly, the majority of rainy points occur in NYC, especially lower Manhattan (top row). In the bottom row, we first divide the city into regions of a few miles and then create a probability distribution over the classes that appear in that sector. Then, we cluster these probability distributions into 4 meta-classes/clusters using k-means. Clearly, close by areas of a city

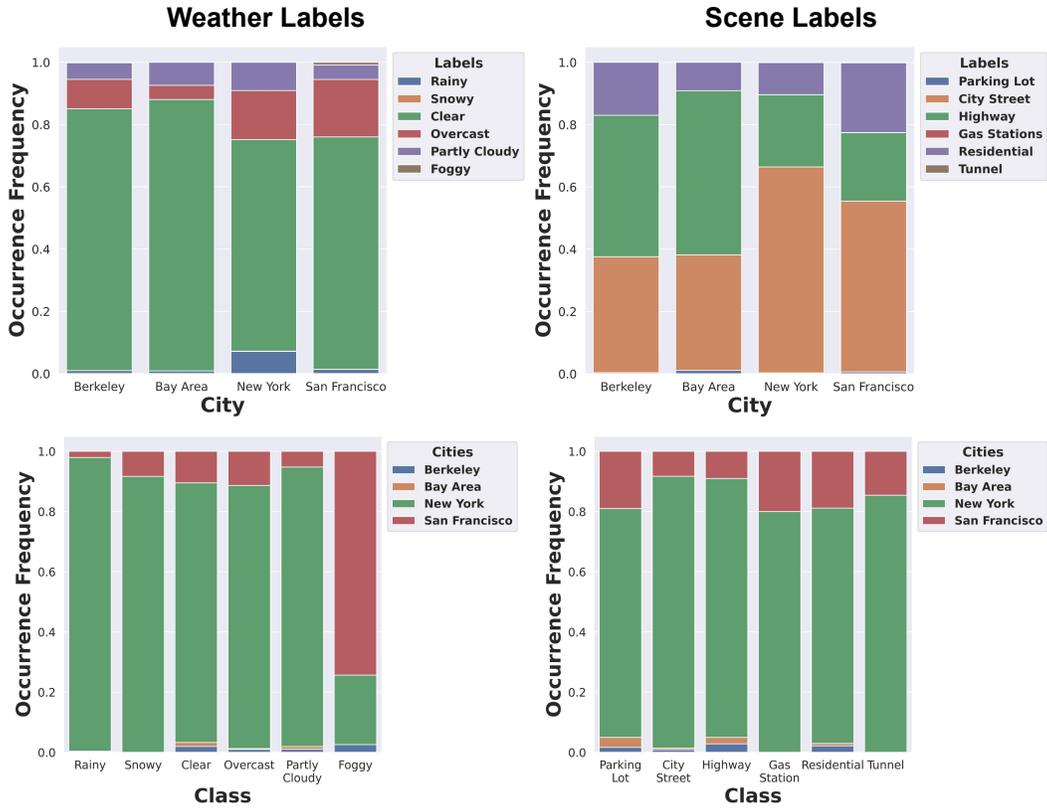


Figure 5: **Heterogeneous Data Label Distributions Across Cities.** Even if we group across a full city, the cities differ in their label distributions. For example, a majority of scenes with rain occur in New York (left), while a majority of scenes with fog occur in SF. Thus, this paper’s algorithms to coordinate data collection in heterogenous environments are needed for real-world AV datasets.

have similar probability distributions over classes, but they are quite distinct in different geographic areas.

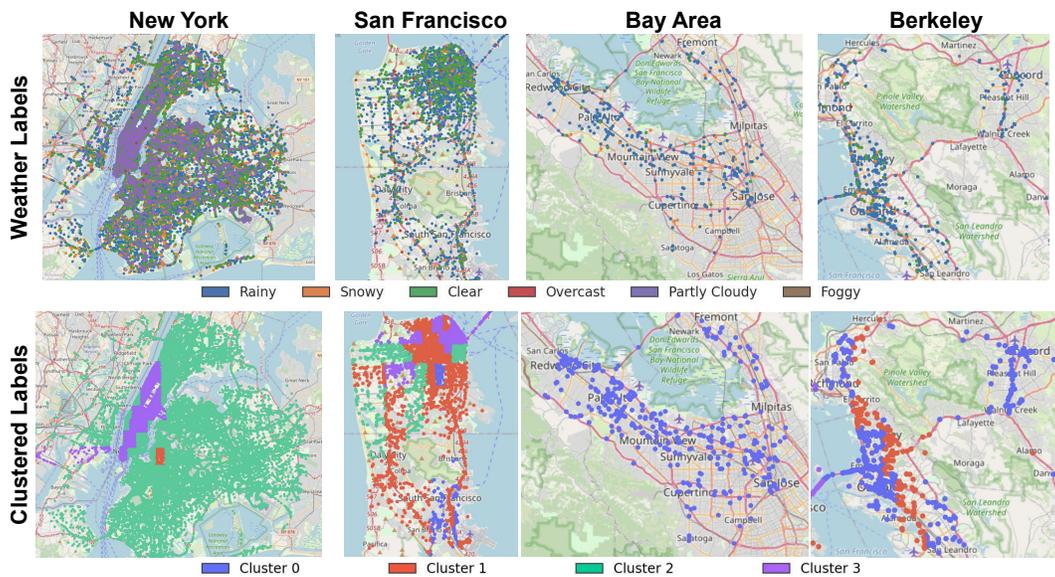


Figure 6: **Heterogeneous data in real world AV datasets.** Top: The distribution of weather across cities is skewed, with much of the rain in the DeepDrive dataset in New York. Bottom: First, we group regions in a city and obtain a frequency distribution over classes in that area. Then, we perform k-means clustering with 4 clusters of the frequency distributions. Clearly, the frequency distributions of different weather conditions are similar locally, but very different across regions of a city.

5.4.6 Visualizing Heterogenous Data Distributions Across Space and Time

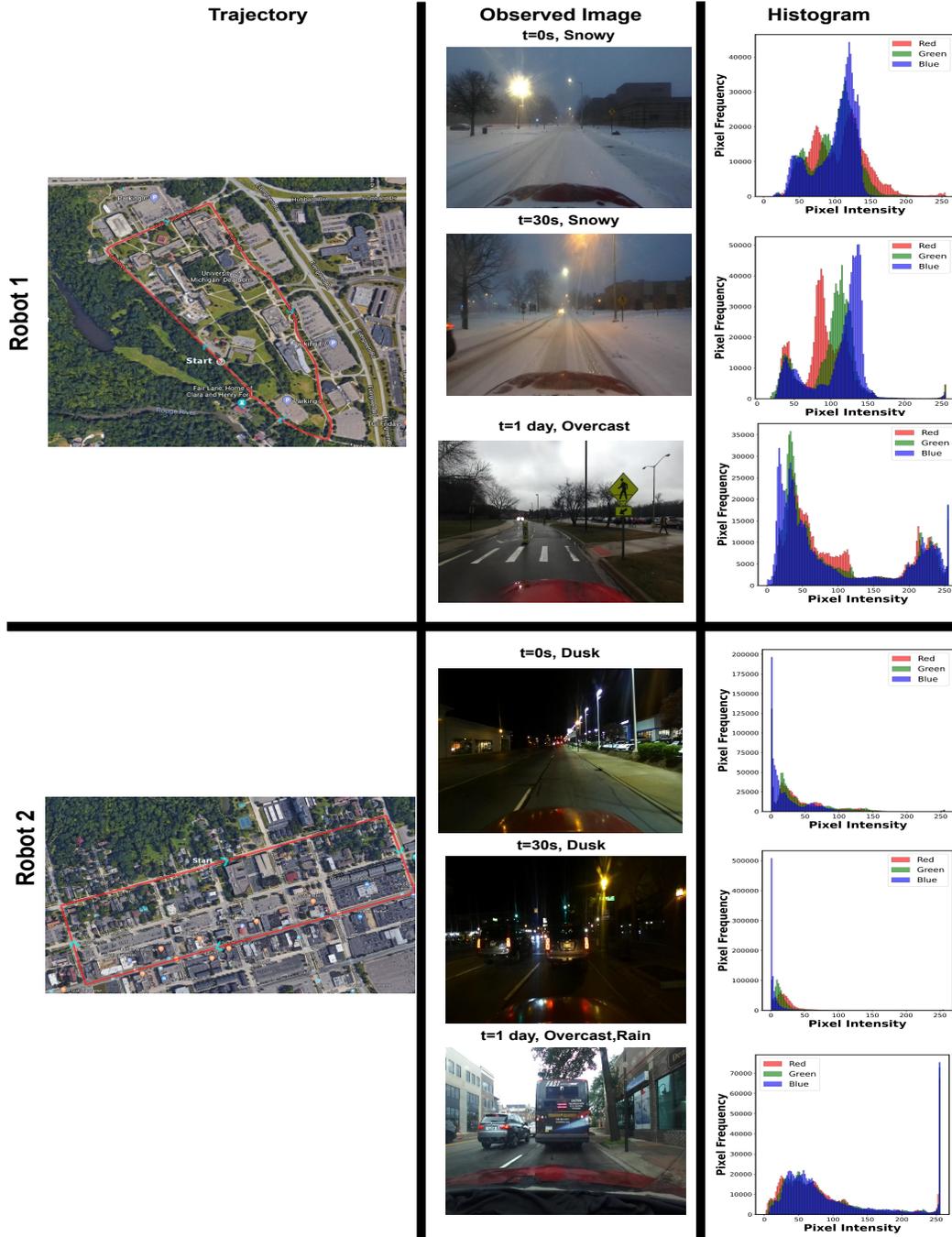


Figure 7: **Observed Image Statistics Differ Across Space and Time for Real-World Robot Trajectories.** We show two robots' trajectories on a map (only 2 for visual clarity). The robots operate in different parts of a city and observe different images - Robot 1 observes snowy and overcast images whereas Robot 2 observes Dusk and Overcast/Rain images. For all robots, we see that closely-spaced frames (30 seconds apart) have the same class. But even then, the pixel values are different from each other, as seen by the histogram of pixel intensities being different. For the same robots, randomly selected images from 1 day later have very different classes and pixel value distributions. Also, when we compare different locations we see that the pixel distributions vary significantly.

We now show that the distribution of observed image pixels and classes change for a robot across its trajectory (space and time). Fig. 7 shows this heterogeneity across several real-world trajectories

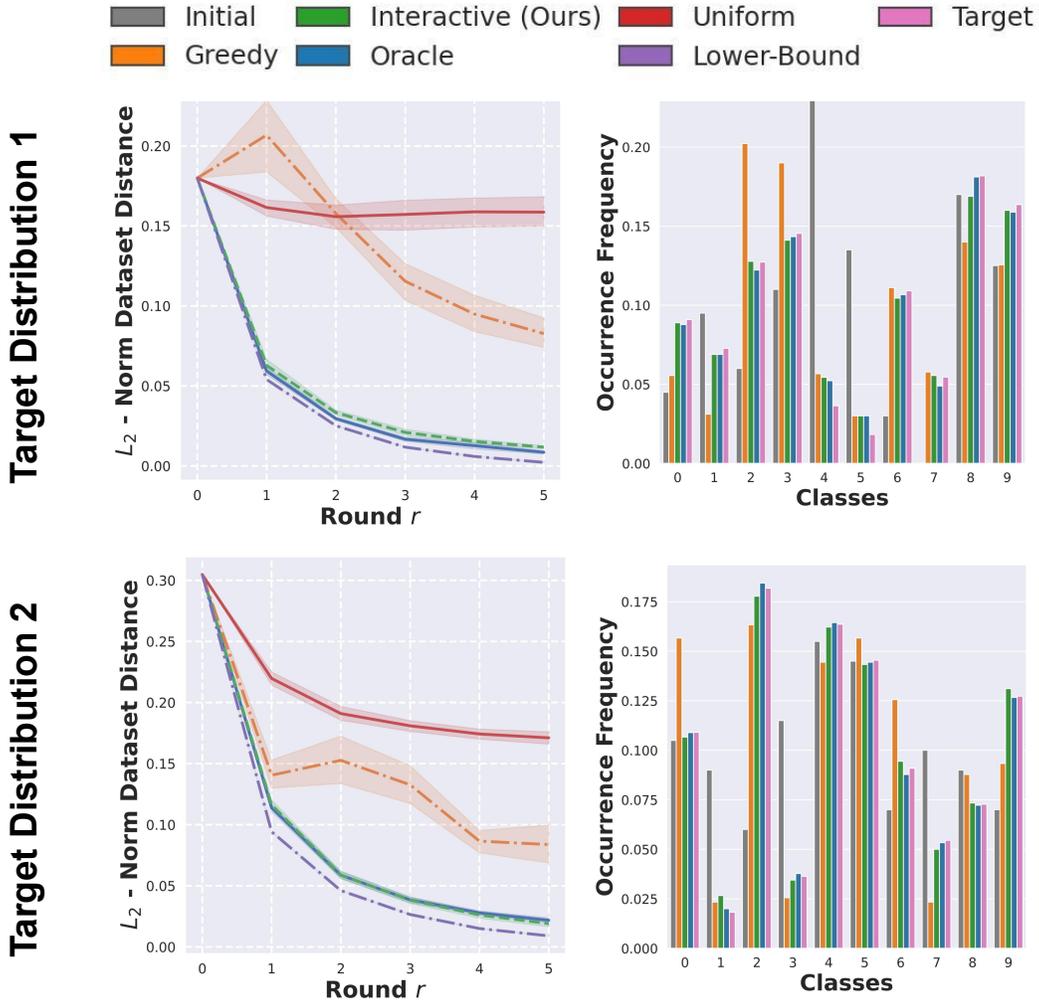


Figure 8: **Our INTERACTIVE policy quickly converges to non-uniform target data distributions.** Each row shows an independent experiment with a different, non-uniform desired target distribution set by the robotist (pink). In the barplot, we see that the initial distribution of classes seen by heterogeneous robots is skewed. Clearly, our Interactive policy (green) quickly converges to the desired non-uniform target data distribution (left panel). Moreover, the *final* data distribution achieved by our Interactive policy (green) closely matches the desired target distribution (pink) in the right barplots. Finally, we significantly outperform the heuristic benchmarks of greedy and uniform sampling. Thus, we verify that our algorithm works even when the target distribution is not uniform but rather any arbitrary target distribution. This is guaranteed by our theory since the target distribution is fixed and we have a convex loss function penalizing the difference between the current cloud distribution and the target (see Theorems 1-3).

on a map. These complement the computed aggregate statistics in Appendix Figs. 4-6. We now describe visualize the heterogenous data distributions robots see in real-world autonomous driving datasets.

Any Given Robot Sees Different Pixels/Classes As it Moves Along its Trajectory (Space + Time) :

Fig. 7 shows two robots’ trajectories on a map (only 2 for visual clarity). The robots operate in two different parts of a city in the Adversarial Weather dataset within different environments. Robot 1 observes mostly snowy and overcast images, whereas Robot 2 observes dusk and overcast images. For any two randomly selected frames close together (30 seconds apart), the images are temporally correlated so are not independent. Further, for the same robot, randomly selected images 1 day later have totally different classes (the scene transitions from snow to overcast or dusk to rain).

Therefore, the classes are not identically distributed across space nor across time. We observed this for hundreds of robots and randomly selected 2 for visual clarity.

Aggregate Statistics Across Real-World Driving Datasets:

Next, we show such heterogeneity across a full dataset. We do not shuffle these datasets artificially - they are the original, naturally-occurring, real world adversarial weather and DeepDrive datasets. For each robot, we show that the distribution of classes it sees across its trajectory is very different from robot to robot in Appendix Figs. 4-6.

Histogram of Pixel Differences:

Our algorithm targets distributed collection of diverse classes of data. Since these robots observe diverse real-world images across space and time, naturally the distribution of raw pixels will be different. To prove this, we compute a histogram of pixel color values in the right panel of Fig. 7. First, in Fig. 7, we show that even for the same robot, the distribution is different for two randomly selected images which are 30 seconds apart. Then, in Fig.7 column 3, we compare the distribution of pixel intensities for different robots and different classes, which are indeed different.

5.4.7 Convergence to Non-Uniform Target Data Distributions

We now illustrate that our algorithm can converge to any non-uniform target distribution. Our algorithm does not assume the “true” distribution of classes is uniform. Instead, we let a roboticist choose *any* desired “target” distribution of classes that they want to collect in the cloud for their analytics or ML use case. Our theory is general – once the target distribution is chosen and fixed, we have a convex loss function between the current and target distribution, which guarantees convergence via Theorems 1-3.

Often, in practice, a roboticist might want an equal distribution of classes to train a robust ML model, which was the case we showed in Figure 3 of the main paper. However, if the roboticist wants to create a model to especially focus on weak points (safety critical examples) for which we have few examples in the cloud, they can select a non-uniform target distribution. Fig. 8 shows that our algorithm is able to easily converge to a non-uniform target distribution. In the barplot, the initial data distribution among robots is highly skewed (grey). The target distribution is pink and is clearly non-uniform. The left panel illustrates our Interactive and Oracle policies quickly converge as our theory guarantees. Moreover, the barplot shows the final data distribution achieved in the cloud under the Interactive and Oracle policies closely matches the target (in pink) and is much closer than the heuristic uniform sampling and greedy sampling benchmarks (orange and red). Finally, we repeat this experiment for another non-uniform target distribution in the bottom panel and see it also converges, as expected by our theory.

5.5 Performance Gap Between ORACLE and GREEDY

Here, we prove the performance gap between ORACLE and INTERACTIVE. All definitions with *feasible* mean that they satisfy the constraints in Eq. 1, 2, and 3.

Definition 3 (Feasible space of all robots). *A feasible space of all robots under ORACLE is the Minkowski sum of all robots' feasible spaces (see Def. 2). The feasible space of all robots is:*

$$H^r = \left\{ \sum_{i=1}^{N_{robot}} v_i^r \mid \forall i = 1, \dots, N_{robot}, v_i^r \in H_i^r \right\}.$$

Definition 4 (Feasible actions). *We define the optimal feasible action for robot i in round r as $v_i^{*,r}$. The symbol of $*$ can denote g or o , standing for GREEDY or ORACLE respectively. Also, we define the optimal action $a_i^{*,r}$ with the left inverse of P_i^r as $P_i^{r\dagger}$. The actions are obtained by solving the optimization problems under different scenarios like ORACLE or GREEDY as follows:*

$$\begin{aligned} v_i^{g,r} &= \arg \min_{v_i^r} \mathcal{L}(\rho_{\mathcal{D}_c} + v_i^r, \rho_{\mathcal{D}_{target}}). \\ \text{subject to: } & v_i^r \in H_i^r \\ v_i^{o,r} &= \arg \min_{v_i^r} \mathcal{L}(\rho_{\mathcal{D}_c} + \sum_{i=1}^{N_{robot}} v_i^r, \rho_{\mathcal{D}_{target}}). \\ \text{subject to: } & v_i^r \in H_i^r, \forall i = 1, \dots, N_{robot} \\ a_i^{g,r} &= P_i^{r\dagger} v_i^{g,r}. \\ a_i^{o,r} &= P_i^{r\dagger} v_i^{o,r}. \end{aligned}$$

Now, we compare the optimal values of ORACLE and GREEDY and show that ORACLE outperforms GREEDY. Then we formulate the performance bound between ORACLE and GREEDY with a lower bound.

Definition 5 (Optimal values of loss functions). *For simplicity, we define the optimal values of loss functions under ORACLE and GREEDY policies as \mathcal{L}^g and \mathcal{L}^o . These are the values of the loss functions resulting from feasible actions:*

$$\begin{aligned} \mathcal{L}^{g,r} &= \mathcal{L}(\rho_{\mathcal{D}_c} + \sum_{i=1}^{N_{robot}} v_i^{g,r}, \rho_{\mathcal{D}_{target}}), \\ \mathcal{L}^{o,r} &= \mathcal{L}(\rho_{\mathcal{D}_c} + \sum_{i=1}^{N_{robot}} v_i^{o,r}, \rho_{\mathcal{D}_{target}}). \end{aligned}$$

Theorem 4 (ORACLE outperforms GREEDY). *The optimal value of GREEDY policy $\mathcal{L}^{g,r}$ is always greater than or equal to the optimal value of ORACLE policy $\mathcal{L}^{o,r}$, i.e. $\mathcal{L}^{g,r} \geq \mathcal{L}^{o,r}$.*

Proof. By Def. 3 and 4, $\sum_{i=1}^{N_{robot}} v_i^{g,r} \in H^r$ and $\sum_{i=1}^{N_{robot}} v_i^{o,r} \in H^r$. By Def. 5, \mathcal{L}^o is the minimum of the loss function under feasible space H^r , so all other loss functions generated by vectors in the same feasible space must be larger. Therefore, $\mathcal{L}^{g,r} \geq \mathcal{L}^{o,r}$. \square

Theorem 5 (Performance gap of ORACLE and GREEDY). *We use Def. 5 and the triangle inequality to show the performance gap between ORACLE and GREEDY.*

$$\begin{aligned} 0 \leq \mathcal{L}^{g,r} - \mathcal{L}^{o,r} &= \|\rho_{\mathcal{D}_{target}} - \rho_{\mathcal{D}_c} - \sum_{i=1}^{N_{robot}} v_i^{g,r}\| - \|\rho_{\mathcal{D}_{target}} - \rho_{\mathcal{D}_c} - \sum_{i=1}^{N_{robot}} v_i^{o,r}\| \\ &\leq \left\| \sum_{i=1}^{N_{robot}} (v_i^{o,r} - v_i^{g,r}) \right\| \end{aligned}$$

As a special illustrative case, if all H_i^r 's are identical, then $v_i^{g,r} = v_i^{o,r}$. According to Thm. 5, $0 \leq \mathcal{L}^{g,r} - \mathcal{L}^{o,r} \leq 0$, hence $\mathcal{L}^{g,r} - \mathcal{L}^{o,r} = 0$. GREEDY is the optimal policy in this case, and there is no need to do cooperative data sharing. This could arise, for example, when all robots have the same vision model uncertainty and same local data distribution.

Next, we show an easy way to obtain the lower bound of ORACLE, using the Euclidean norm as an example. We create a new relaxation of Eq. 1 by removing the first constraint in Eq. 1. That is, the number of the data-points uploaded need not be positive. In this case, since robots can upload *negative* data-points, any combination of data-point is feasible as long as its sum is less than or equal to N_{cache} . Thus, confusion matrices of robots do not matter here, and this mimics a case with no perceptual uncertainty.

Lemma 6 (Lower bound of ORACLE). *The relaxation of Eq. 1 by removing the first constraint in Eq. 1 is the lower bound of ORACLE.*

Proof. The original feasible set of the optimization problem is a subset of the new feasible set since we expand the set by removing a constraint from the original problem. Hence, we know the new optimal value is less than or equal to the original one. Namely,

$$\begin{aligned} \mathcal{L}^{low,r} = \min \quad & \mathcal{L}(\rho_{\mathcal{D}_c^r} + \sum_{i=1}^{N_{\text{robot}}} v_i^r, \rho_{\mathcal{D}_{\text{target}}}) \leq \mathcal{L}^{o,r}. \\ \text{subject to:} \quad & \mathbf{1}^T \cdot v_r^i \leq N_{\text{cache}}; \quad \forall i = 1, \dots, N_{\text{robot}} \end{aligned}$$

For L_2 norm, a closed-form solution of $\mathcal{L}^{low,r}$ can be obtained by projecting the objective value to the feasible space:

$$\mathcal{L}^{low,r} = \max(\mathbf{1}^T (\rho_{\mathcal{D}_{\text{target}}} - \rho_{\mathcal{D}_c^r}) - N_{\text{cache}} \times N_{\text{robot}}, 0) \times \sqrt{N_{\text{class}}}.$$

□

5.6 Theorem 1: While loop in Alg. 1 converges eventually

We first show that there is a unique solution of INTERACTIVE then show that Alg. 1 will converge to that solution.

Lemma 7 (Uniqueness of INTERACTIVE solution). *The optimal solution of INTERACTIVE $\sum_{i=1}^{N_{\text{robot}}} v_i^{int,r}$ is unique.*

Proof. We use proof by contradiction. First, we know $\sum_{i=1}^{N_{\text{robot}}} v_i^{int,r} \in H^r$, and H^r is a convex set. If there exist more than two optimal solutions, we arbitrarily pick two of them and name them $v^{int,r}$ and $v'^{int,r}$. Since $\mathcal{L}(\cdot, \cdot)$ is strictly convex,

$$\begin{aligned} \mathcal{L}(\rho_{\mathcal{D}_c^r} + \frac{1}{2}[v^{int,r} + v'^{int,r}], \rho_{\mathcal{D}_{\text{target}}}) &< \frac{1}{2}[\mathcal{L}(\rho_{\mathcal{D}_c^r} + v^{int,r}, \rho_{\mathcal{D}_{\text{target}}}) + \mathcal{L}(\rho_{\mathcal{D}_c^r} + v'^{int,r}, \rho_{\mathcal{D}_{\text{target}}})] \\ &= \mathcal{L}(\rho_{\mathcal{D}_c^r} + v^{int,r}, \rho_{\mathcal{D}_{\text{target}}}). \end{aligned}$$

Then $\frac{1}{2}[v^{int,r} + v'^{int,r}]$ achieves a lower loss function and contradicts with our assumption that $v^{int,r}$ and $v'^{int,r}$ are optimal solutions. Hence, $\sum_{i=1}^{N_{\text{robot}}} v_i^{int,r}$ is unique. □

Theorem (Convergence Eventually). *The while loop (lines 6 - 11) in Alg. 1 will eventually converge.*

Proof. For the proof of convergence, refer to Theorem 2 of [41]. The potential function defined in [41] corresponds to the negative value of our objective function, as stated in section 5.1. The random revision law there is replaced by our deterministic order of updates in line 7. From Lemma 7, we know there is only one unique solution, thus eventually Alg. 1 will converge to it. □

Intuitively, a potential game with a strictly concave potential function will converge eventually since all players (robots in our case) strictly increase the potential function.

5.7 Theorem 2: INTERACTIVE converges to ORACLE

We prove our proposed method INTERACTIVE described in Eq. 3 and Alg. 1 is equivalent to ORACLE as described in Thm. 2. We discuss two cases respectively: $\mathbf{1}^\top(\rho_{\mathcal{D}_{\text{target}}} - \rho_{\mathcal{D}_c^r}) > N_{\text{robot}} \times N_{\text{cache}}$ and $\mathbf{1}^\top(\rho_{\mathcal{D}_{\text{target}}} - \rho_{\mathcal{D}_c^r}) \leq N_{\text{robot}} \times N_{\text{cache}}$. The first case holds for all rounds except for the last round that reaches the target distribution, upon which data collection terminates (see Thm. 3). When $\mathbf{1}^\top(\rho_{\mathcal{D}_{\text{target}}} - \rho_{\mathcal{D}_c^r}) > N_{\text{robot}} \times N_{\text{cache}}$ holds, INTERACTIVE will certainly converge to ORACLE in one while loop execution (running Alg. 1 line 6 - 11 once). While in the last round, $\mathbf{1}^\top(\rho_{\mathcal{D}_{\text{target}}} - \rho_{\mathcal{D}_c^r}) \leq N_{\text{robot}} \times N_{\text{cache}}$ holds, and it takes more than one execution to converge.

Lemma 8 (Uniqueness of ORACLE solution). *The optimal feasible action of ORACLE, namely $\sum_{i=1}^{N_{\text{robot}}} v_i^{o,r}$, is unique.*

Proof. The proof is similar to Lemma 7. We use proof by contradiction. First, we know $\sum_{i=1}^{N_{\text{robot}}} v_i^{o,r} \in H^r$, and H^r is a convex set. If there exist more than two optimal solutions, we arbitrarily pick two of them and name them v and $v^{o,r}$. Since the loss $\mathcal{L}(\cdot, \cdot)$ is a strictly convex function,

$$\begin{aligned} \mathcal{L}(\rho_{\mathcal{D}_c^r} + \frac{1}{2}[v^{o,r} + v'^{o,r}], \rho_{\mathcal{D}_{\text{target}}}) &< \frac{1}{2}[\mathcal{L}(\rho_{\mathcal{D}_c^r} + v^{o,r}, \rho_{\mathcal{D}_{\text{target}}}) + \mathcal{L}(\rho_{\mathcal{D}_c^r} + v'^{o,r}, \rho_{\mathcal{D}_{\text{target}}})] \\ &= \mathcal{L}(\rho_{\mathcal{D}_c^r} + v^{o,r}, \rho_{\mathcal{D}_{\text{target}}}). \end{aligned}$$

Then $\frac{1}{2}[v^{o,r} + v'^{o,r}]$ achieves a lower loss function and contradicts with our assumption that $v^{o,r}$ and $v'^{o,r}$ are optimal solutions. Hence, $\sum_{i=1}^{N_{\text{robot}}} v_i^{o,r}$ is unique. \square

Theorem (INTERACTIVE converges to ORACLE). *The while loop in Alg. 1 line 6 - 11 is guaranteed to return action $a_i^{\text{int},r}$ that is equal to the ORACLE policy's action, $a_i^{o,r}$. $a_i^{\text{int},r}$ denotes the action of robot i at the end of round r using the INTERACTIVE policy. Similarly, $a_i^{o,r}$ denotes the action of ORACLE policy.*

Proof. The convergence (optimality) conditions for the convex optimization problems of all robots are of this form with the gradient of the loss function $\nabla_{v_i^{\text{int},r*}} \|\rho_{\mathcal{D}_{\text{target}}} - \rho_{\mathcal{D}_c^r} - \sum_{j=1}^{N_{\text{robot}}} v_j^{\text{int},r*}\|$:

$$\begin{aligned} \forall i, v_i \in H_i^r, \\ (v_i - v_i^{\text{int},r*})^\top \nabla_{v_i^{\text{int},r*}} \|\rho_{\mathcal{D}_{\text{target}}} - \rho_{\mathcal{D}_c^r} - v_i^{\text{int},r*} - \sum_{i \neq j, j=1}^{N_{\text{robot}}} v_j^{\text{int},r*}\| \\ = (v_i - v_i^{\text{int},r*})^\top \nabla_{v_i^{\text{int},r*}} \|\rho_{\mathcal{D}_{\text{target}}} - \rho_{\mathcal{D}_c^r} - \sum_{j=1}^{N_{\text{robot}}} v_j^{\text{int},r*}\| \geq 0. \end{aligned}$$

By the chain rule,

$$\nabla_{v_i^{\text{int},r*}} \|\rho_{\mathcal{D}_{\text{target}}} - \rho_{\mathcal{D}_c^r} - \sum_{j=1}^{N_{\text{robot}}} v_j^{\text{int},r*}\| = \nabla_{\sum_{j=1}^{N_{\text{robot}}} v_j^{\text{int},r*}} \|\rho_{\mathcal{D}_{\text{target}}} - \rho_{\mathcal{D}_c^r} - \sum_{j=1}^{N_{\text{robot}}} v_j^{\text{int},r*}\|.$$

Thus, summing up the optimality conditions of all robots, we get:

$$\begin{aligned} \sum_{i=1}^{N_{\text{robot}}} (v_i - v_i^{\text{int},r*})^\top \nabla_{v_i^{\text{int},r*}} \|\rho_{\mathcal{D}_{\text{target}}} - \rho_{\mathcal{D}_c^r} - \sum_{j=1}^{N_{\text{robot}}} v_j^{\text{int},r*}\| \\ = (\sum_{i=1}^{N_{\text{robot}}} v_i - \sum_{i=1}^{N_{\text{robot}}} v_i^{\text{int},r*})^\top \nabla_{\sum_{j=1}^{N_{\text{robot}}} v_j^{\text{int},r*}} \|\rho_{\mathcal{D}_{\text{target}}} - \rho_{\mathcal{D}_c^r} - \sum_{j=1}^{N_{\text{robot}}} v_j^{\text{int},r*}\| \geq 0. \end{aligned}$$

This implies the optimality condition of ORACLE is:

$$\forall \sum_{i=1}^{N_{\text{robot}}} v_i \in H^r, \left(\sum_{i=1}^{N_{\text{robot}}} v_i - \sum_{i=1}^{N_{\text{robot}}} v_i^{o,r} \right)^\top \nabla_{\sum_{j=1}^{N_{\text{robot}}} v_j^{o,r}} \|\rho_{\mathcal{D}_{\text{target}}} - \rho_{\mathcal{D}_c^r} - \sum_{j=1}^{N_{\text{robot}}} v_j^{o,r}\| \geq 0.$$

We know there is only one unique solution of ORACLE from Lemma 8, so INTERACTIVE will converge to ORACLE in Alg. 1 line 6 - 11, and

$$a_i^{int,r} = a_i^{o,r} = P_i^{r\dagger} v_i^{o,r}.$$

□

Lemma 9 (Sum of feasible actions lies on a hyperplane). *For ORACLE and GREEDY, the sum of action lies on the same hyperplane $\mathbf{1}^\top v = N_{robot} \times N_{cache}$ when $\mathbf{1}^\top (\rho_{\mathcal{D}_{target}} - \rho_{\mathcal{D}_c^r}) > N_{robot} \times N_{cache}$.*

Proof. Since $\rho_{\mathcal{D}_{target}}$ lies outside H^r , the closest point to it must lie on the boundary of the convex set. Thus, $\sum_{i=1}^{N_{robot}} v_i^{o,r}$ lies at the edge of H^r , the hyperplane $\mathbf{1}^\top v = N_{robot} \times N_{cache}$. Thus,

$$\mathbf{1}^\top \sum_{i=1}^{N_{robot}} v_i^{o,r} = N_{robot} \times N_{cache}.$$

Every shared action in Alg. 1 line 5 is the GREEDY action $a_i^{g,r}$ and the corresponding feasible action $v_i^{g,r}$ lies at the edge of H_i^r , the hyperplane $\mathbf{1}^\top v = N_{cache}$ for the same reason as above. Thus, we know:

$$\mathbf{1}^\top \sum_{i=1}^{N_{robot}} v_i^{g,r} = \sum_{i=1}^{N_{robot}} \mathbf{1}^\top v_i^{g,r} = N_{robot} \times N_{cache}$$

The sum of greedy feasible actions also lies on the same hyperplane $\mathbf{1}^\top v = N_{robot} \times N_{cache}$. □

Now, using the fact that the sum of feasible actions lies on the same hyperplane from Lemma 9, we can show that the while loop in Alg. 1 line 6 - 11 will terminate in one iteration.

5.8 Theorem 3: While loop converges in one iteration

Theorem (Convergence in one iteration). *For cases when the total number of uploadable data-points is less than the difference between target cloud dataset $\rho_{\mathcal{D}_{target}}$ and current cloud dataset $\rho_{\mathcal{D}_c^r}$, namely $\mathbf{1}^\top (\rho_{\mathcal{D}_{target}} - \rho_{\mathcal{D}_c^r}) > N_{robot} \times N_{cache}$, the while loop in Alg. 1 line 6 - 11 will terminate in one iteration.*

Proof. Since the optimal solution of ORACLE is unique from Lemma 8, we know the update direction of solution (the vector from the previous solution pointing to the new solution) in the first optimization execution in line 6 - 11 is the vector pointing from the GREEDY feasible solution $\sum_{i=1}^{N_{robot}} v_i^{g,r}$ to the ORACLE solution $\sum_{i=1}^{N_{robot}} v_i^{o,r}$. Both points lie on the hyperplane $\mathbf{1}^\top v = N_{robot} \times N_{cache}$ by Lemma 9. Also, all feasible spaces in Eq. 3 intersect with the hyperplane $\mathbf{1}^\top v = N_{robot} \times N_{cache}$, so all update directions in line 6 - 11 during the while loop lie on the same hyperplane until the solutions converge.

Let the solution after the first iteration of the while loop be v_r^{iter} and the solutions of each for loop execution before it be

$$v_r^{for,j}, \text{ for } j = 1, \dots, N_{robot}.$$

Note that,

$$\begin{aligned} \sum_{i=1}^{N_{robot}} v_i^{o,r} &\in \{v : \mathbf{1}^\top v = N_{robot} \times N_{cache}\}, \\ v_{iter} &\in \{v : \mathbf{1}^\top v = N_{robot} \times N_{cache}\}, \\ v_r^{for,j} &\in \{v : \mathbf{1}^\top v = N_{robot} \times N_{cache}\}, \text{ for } j = 1, \dots, N_{robot}, \end{aligned}$$

since all the updates happen on the hyperplane.

We then assume v_{iter} is not the solution of ORACLE, $\sum_{i=1}^{N_{robot}} v_i^{o,r}$, and prove it is wrong by contradiction. If they are not identical, let the difference between solutions of ORACLE and the first iteration be

$$\Delta v = \sum_{i=1}^{N_{robot}} v_i^{o,r} - v_{iter} \neq 0.$$

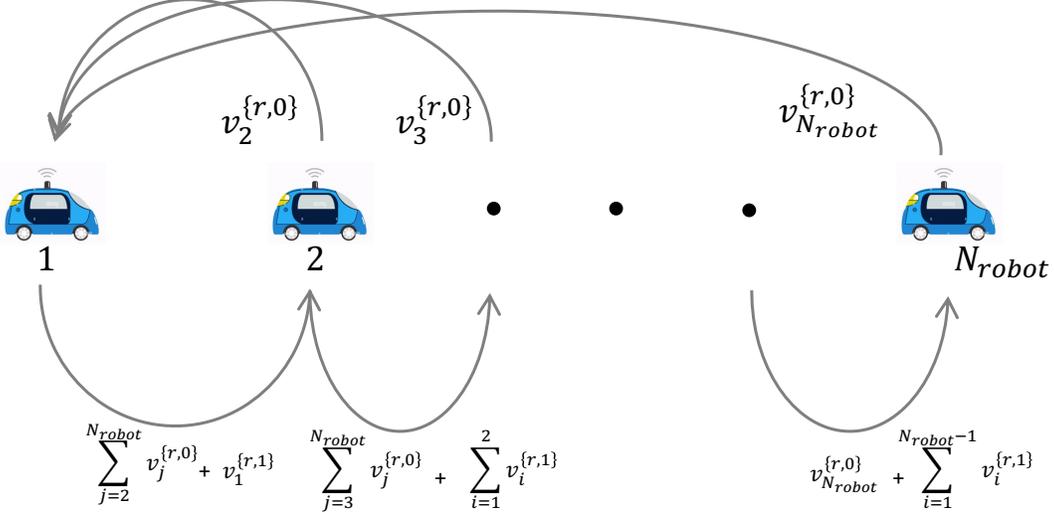


Figure 9: **Communication Optimization in Alg. 1 While loop:** First, each robot shares its greedy actions (grey arrows facing left) $v_i^{r,0}$. Then, each robot passes the *sum* of optimized actions $v_i^{r,1}$ and other robots' actions $v_j^{r,0}$ as opposed to *individual* actions, leading to $O(N_{\text{robot}})$ messages.

Δv is the same direction as all update directions in line 6 - 11. All $v_r^{\text{for},j} + \alpha \Delta v$ are infeasible ($\notin H_r^f$) for any j and an arbitrary small step size of update $\alpha > 0$ because all $v_r^{\text{for},j}$ are already optimal solutions that cannot move further in the update directions. Hence,

$$\sum_{i=1}^{N_{\text{robot}}} v_i^{o,r} = v_{\text{iter}} + \Delta v \notin H^f.$$

This contradicts with Def. 4, so we prove that $\Delta v = 0$ and $\sum_{i=1}^{N_{\text{robot}}} v_i^{o,r} = v_{\text{iter}}$. In other words, the while loop in Alg. 1 line 6 - 11 will terminate in one iteration. \square

5.9 Proposition 1: The total number of messages passed between the robots.

We first calculate the number of messages passed in every iteration of Alg. 1 using an *un-optimized* method of communication that requires $O(N_{\text{robot}}^2)$ messages. Then, we show a simple, optimized method that requires only $O(N_{\text{robot}})$ messages per loop.

Proposition 1 (Total Number of Messages). *The total number of messages passed between the robots in line 5 will be $N_{\text{robot}}^2 - N_{\text{robot}}$. While in each iteration (for loop line 7 - 10), the number is also $N_{\text{robot}}^2 - N_{\text{robot}}$.*

Proof. Each robot i shares its decision $P_i^r a_i^r$ with $(N_{\text{robot}} - 1)$ other robots, and this process repeats N_{robot} times for all robots. Hence, the total numbers of messages passed between the robots in line 5 and for loop line 7 - 10 are both

$$N_{\text{robot}} \times (N_{\text{robot}} - 1) = N_{\text{robot}}^2 - N_{\text{robot}}.$$

\square

5.9.1 An Optimized Method with only $O(N_{\text{robot}})$ messages

Our key insight to reduce communication, shown in Fig. 9, is that robots only need to share their individual actions initially and afterwards can only share *sums* of their actions with each other.

As shown in Fig. 9, let us denote a feasible action on round r by $v_i^r = P_i^r a_i^r$. Further, let us index iterations of communication *within* a loop by k , meaning $v_i^{r,0}$ is the *initial greedy* action from robot

i at round r (i.e., at iteration 0). After solving Prob. 3 once and multiplying by P_i^r , the next action is given by $v_i^{r,1}$. As shown in Fig. 9, all robots send their initial greedy action $v_j^{r,0}$ to robot 1 for $j = 2 \dots N_{\text{robot}}$. This amounts to $N_{\text{robot}} - 1$ messages sent. Then, robot 1 solves Prob. 3, assuming all other robots' actions are fixed, to generate $v_1^{r,1}$. The sum of the new optimized action $v_1^{r,1}$ and previous unoptimized actions $\sum_{j=2}^{N_{\text{robot}}} v_j^{r,0}$ is sent to robot 2. Robot 2 then subtracts its current greedy action $v_2^{r,0}$ in Eq. 3d and solves Prob. 3 again. The process repeats until we reach robot N_{robot} , leading to another $N_{\text{robot}} - 1$ messages. As such, for each while loop iteration, we only need $(N_{\text{robot}} - 1) + (N_{\text{robot}} - 1) = 2(N_{\text{robot}} - 1)$ messages, so $O(N_{\text{robot}})$ messages as opposed to $O(N_{\text{robot}}^2)$.