Scalable Nonparametric Bayesian Learning for Dynamic Velocity Fields (Supplementary Material)

Sunrit Chakraborty[*1] **Aritra Guha** [2] **Rayleigh Lei**[3] **XuanLong Nguyen**[1]

[1]Department of Statistics, University of Michigan, Ann Arbor, MI, USA
[2]Data Science & AI Research, Chief Data Office, AT&T, Bedminster, NJ, USA
[3]Department of Statistics, University of Washington, Seattle, WA, USA

A PRELIMINARIES

Here, we discuss Hidden Markov model briefly and provide additional details about the infinite HMM.

A.1 HMM AND VITERBI ALGORITHM

Hidden Markov model is a statistical model used for time series data, where the observations are believed to depend on some underlying unobserved states, which form a Markov chain. The number of distinct values taken by the state variables is assumed known, say K. This structure allows temporal dependence among the observations and is one of the most powerful unsupervised learning tools that has applications in statistics, economics, bioinformatics, thermodynamics, statistical physics among others. Assuming the observations are denoted by $\{X_t\}$ and the latent states by $\{z_t\}$ which take values in $[K]$, the model has 3 main components, the initial distribution π_0, which is the distribution of z_1; the transition matrix Π, which dictates the Markov dynamics of the states, i.e. $\pi_{ij} = p(z_2 = j|z_1 = i)$ and the emission probabilities $p(x|z = k)$, which depends on the type of observations. Example of emission include multinomial for discrete observations and Gaussian or Poisson for continuous observations.

Inference in such models is involved because of the complex temporal dependence between the variables. Dynamic programming allows one to compute likelihood for such models using message passing. Baum-Welch algorithm is the most popular method for estimating the parameters in such models, which is a special case of the Expectation Maximization algorithm used to compute MLE in latent variable models.

Another related problem is given the parameters, what sequence of states maximize the likelihood. This decoding problem is also solved efficiently using dynamic programming, which results in the so called Viterbi algorithm.

A.2 INFINITE HMM

One of the principle drawbacks of HMM is that the number of states need to be fixed. In some applications, this might not be a big problem since there is some understanding of what the states might represent and hence, to narrow down the possible number of states to use. However, in some applications, this is not the case and it would lead to fitting several HMM with different values of K and comparing them by either prediction accuracy or penalized likelihood. However, with large datasets, it is extremely inefficient to do this and as a result, a method which does not fix K but estimates it from the data might seem more appropriate. Moreover, in certain applications, the number of states might keep increasing with time. Each of the latent states represent a type of homogeneity among observations at different time points with the same state value, thus each state represents a particular pattern of the observed data. In traffic data, each state represents a particular traffic pattern and the number of patterns might increase with increasing total observation time, since new patterns might emerge.

[*]Corresponding author. Email: sunritc@umich.edu

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

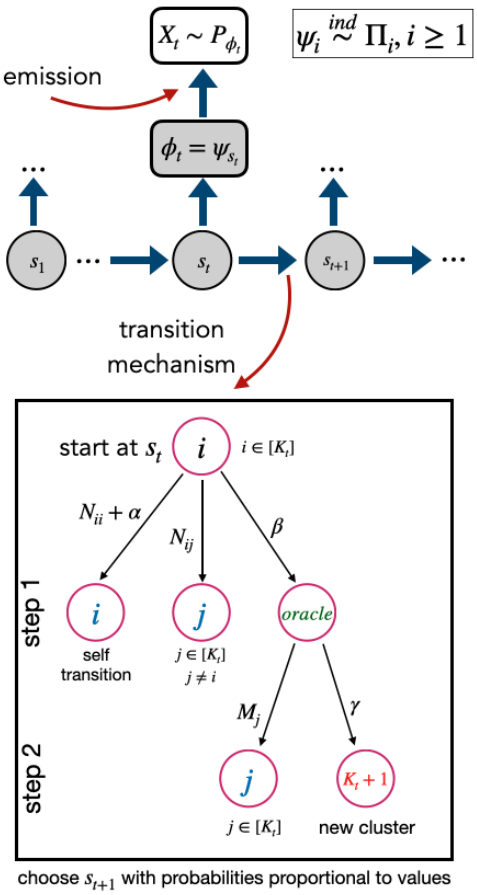


Figure 1: Illustration of iHMM Model.

This brings us to the infinite HMM model, a Bayesian nonparametric model which allows countably infinite number of components, thus allowing the model to estimate the number of distinct states from the data. We briefly describe the infinite HMM prior structure. Given parameters (α, β, γ), we describe the prior using the generative model. To draw a sample of states $\{s_t\}$ from this model, we start by setting $s_1 = 1$. We also keep a sequence of binary variables, called oracle variables, denoted by o_t, which we initialize as $o_1 = 1$.

Having generated $\{s_{1:t}\}$ and $\{o_{1:t}\}$, let K_t be the number of distinct elements in $\{s_{1:t}\}$ and N and M denote the transition count matrix and oracle vector respectively, as defined in the main text. The conditional transition probabilities $p(s_{t+1}|s_{1:t}, o_{1:t})$ are given in Eq (10) and (11) in the main paper. Figure 1 shows this transition process. It is essentially a two-step mechanism, where in the first step, the state can either stay in one of the explored states (this would give $o_{t+1} = 0$) or move to an oracle (whence $o_{t+1} = 1$). In this case, in step 2 the process might then again go to one of the explored states or might transition to an unexplored state, thereby starting a new cluster. The two steps can be thought of as two allocators: a local allocator which assigns the next time point to one of the previously explored states and a global allocator which in addition to one of the existing states, can allocate the next time point to a new state. The oracle variable decides which allocator is used at that time point. The parameter α controls the amount of self-transitions, i.e. going from $s_t = i$ to $s_{t+1} = i$, β controls the amount of oracle use, while γ controls the tendency of the process to explore new clusters. We write $s_t \sim \text{iHMM}(\alpha, \beta, \gamma)$, to indicate that we place this infinite HMM as the prior on $\{s_t\}$.

B COMPUTATION OF SPARSE GP IN FORWARD PASS

Consider a fixed set of inducing points $\boldsymbol{Z}$ of size m. Using notations from Section 4 in the main paper, whenever a new cluster is found, we update the variational parameters μ_m and Σ_m using Equations (7) and (8). Then, during the forward pass step at time t, to compute $p(\mathcal{D}_{t+1}|s_{1:(t+1)}, \mathcal{D}_{1:t})$, we still use Equation (15), but for $\mu_t^{(k)}$ and $\Sigma_t^{(k)}$, instead of using

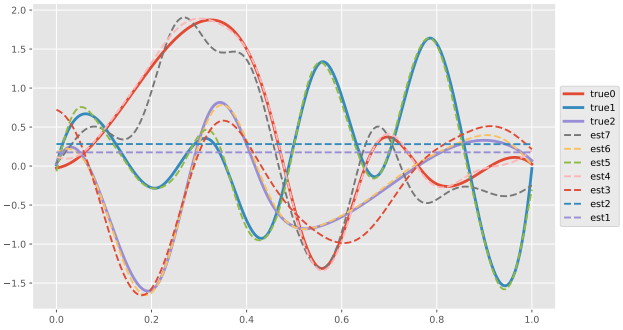


Figure 2: An example of estimated clusters after forward pass with $K = 3$ true functions (solid) in $P = D = 1$, dashed functions are the posterior means of the fitted GPs for each of 7 identified clusters.

Equation (2), we use Equations (6) and (5), which does not require recomputation if $K_{\theta_k}(\boldsymbol{Z})^{-1}$, which is much smaller in size, is stored (all referenced equations are from main paper). Once $\hat{s}_{t+1} = k$ is chosen, either initialize a new SGP (if $k = K_t + 1$) or update the variational parameters of the chosen existing SGP using the current data, which involves inverting a $m \times m$ matrix for all time points t. The only other heavy computation is $K(\boldsymbol{Z}, \boldsymbol{X}^{(k)}_{1:(t+1)})K(\boldsymbol{Z}, \boldsymbol{X}^{(k)}_{1:(t+1)})^\top$ (comes up in Eq (9)), which can be sped up with the following observation. Since $\boldsymbol{X}^{(k)}_{1:(t+1)} = (\boldsymbol{X}^{(k)}_{1:t}, \boldsymbol{X}_{t+1})$,

$$K\left(\boldsymbol{Z}, \boldsymbol{X}^{(k)}_{1:(t+1)}\right)K(\boldsymbol{X}^{(k)}_{1:(t+1)}, \boldsymbol{Z}\Big)$$
$$= K(\boldsymbol{Z}, \boldsymbol{X}^{(k)}_{1:t})K(\boldsymbol{X}^{(k)}_{1:t}, \boldsymbol{Z}) + K(\boldsymbol{Z}, \boldsymbol{X}_{t+1})K(\boldsymbol{X}_{t+1}, \boldsymbol{Z}).$$

Thus, by storing the appropriate matrices, this step will involve only multiplying $m \times n_t$ matrix with $n_t \times m$, instead of $m \times \tilde{n}^{(k)}_t$ with $\tilde{n}^{(k)}_t \times m$.

C DETAILS ABOUT THE REFINEMENT STEP

After finishing the forward pass, we have estimates of $s_{1:T}$, based on which we have K_T components, each with a SGP. We discuss a few problems that might arise at this stage. To motivate the problems, consider finite dimensional Gaussian distribution emissions with known covariance $\sigma^2 I$ and different unknown means $\mu_k \in \mathbb{R}^n$ (this arises in case where $\boldsymbol{X}_t$ is the same for all t) for simplicity. Because the forward pass is sequential in nature, each cluster initializes somewhere based on the first time when $\mathcal{D}_t$ was assigned to it. This could lead to the following obvious two issues, are the clusters initialised properly and what if $\mathcal{D}_t$ is incorrectly clustered to k because there is insufficient confidence about cluster k (possibly t is small and $\sum_r \mathbf{1}(\hat{s}_r = k)$ is small). Neither of these problems will be sorted by a single forward pass. For example regarding the first issue, it is possible that $\mathcal{D}_t$ corresponding to the first t for which $s_t = k$ for a specific cluster k, might be an outlying point for the distribution $\mathcal{N}(\mu_k, \sigma^2 I)$. Figure 2 shows an example for the case $D = P = 1$, showing the many extraneous clusters identified during the forward pass. We can see it identifies some clusters which are all close to one of the true functions (e.g. *est4* in pink and *est7* in gray are both close to *true0* in red) and others which are pretty random, e.g *est2* in blue (on inspection, the GPs for these clusters have high posterior variance, because a very few time points were associated to it). This leads us to the following refinement step.

We start with the smallest cluster, i.e. $\arg\min \sum_t \mathbf{1}(\hat{s}_t = k)$ and work upwards. We remove this cluster temporarily, i.e. remove its effect from N_T, M_T (to get $N_T^{(-k)}$ and $M_T^{(-k)}$) and remove the kth SGP model, thus bringing down the number of clusters to $K_T - 1$. Now, we use Viterbi algorithm to reassign the states of the points $\mathcal{D}^{(k)}_{1:T}$. However, instead of treating it as a HMM with $K_T - 1$ components, we allow a new or unexplored state as another option, thus using a total of K_T components again. The iHMM gives the transition probabilities $p(s_t = j | s_{t-1} = i, N_T^{(-k)}, M_T^{(-k)})$ for all transitions from existing clusters other than k to themselves and the new cluster, while we use uniform distribution as the transition from the new cluster to any of the existing ones (since the new cluster indicates no prior knowledge of transitions from it to other states). The emission probabilities $p(\mathcal{D}_t | s_t = j)$ are computed as in Equation (15) (main paper) for all existing clusters j, while for the new, we use the marginal likelihood of that time point. Using these, we can perform a Viterbi step to reassign

s_r for all time points r, where $\hat{s}_t = k$ previously. We keep track of the number of times the *new* cluster was chosen. If it exceeds a certain fixed number,n_0, we keep this as a cluster, otherwise we use the results from Viterbi to assign them to the other clusters.

We expect that smaller clusters, created due to incorrect initialization of the corresponding cluster, would be reassigned to the correct one after this. This step reduces the number of clusters by deciding whether a particular cluster is indeed required or is redundant and can be absorbed into one of the others. In context of Figure 2, for the smaller clusters (like blue and yellow, or even the red), since each of the true functions is well represented by one of the estimated clusters, this step would reassign time points associated to these smaller clusters after forward pass to their respective correct ones. By reducing the number of clusters, this gives interpretability to each of the clusters as depicting a distinct pattern.

C.1 USE OF BLOCKS

The estimates of the parameter is heavily influenced by the order in which the training data is collected. For example, if t is the first time point where we have observations from a new cluster, then the initial estimate of this cluster parameters relies on $\mathcal{D}_t$. If this data is not a good representative of the specific cluster (say an outlying point), then this might lead to unstable subsequent decisions. For i.i.d. data, we could have fit this process multiple times by giving the observations a random permutation, however, in our case the temporal order of the data must be maintained. Furthermore, if T is large, then the forward pass and refinement might be slow due to the fact that being a sequential method, it cannot be parallelized. Motivated by the need to have multiple initializations to overcome the issues associated with a specific sequence and speeding up the process, we propose to divide the data into blocks $\mathbf{B}_j = \{\mathcal{D}_t : (j-1)m_0 + 1 \leq t < m_0 j\}$, of size m_0 and perform forward pass and refinement on each of these blocks independently. Running the forward pass and refinement on each block can be parallelized, since they are independent processes.

We discuss how to combine the results obtained from different blocks. For each block we obtain a possibly different number of components, let us denote the kth component from block j as $\hat{f}_{jk}$. We run K-Means on $\{\hat{f}_{jk}(\boldsymbol{Z}) : \forall k, j\}$, where $\boldsymbol{Z}$ was the set of fixed inducing points and based on Silhoutte coefficient, we select the optimal number of overall components K. Then, using the K-means results and state labels obtained for each block, we assign new state labels for every t. For example, say $\mathcal{D}_t$ is the t'-th data point of block j and during the run of $\mathbf{B}_j$, its state was estimated as k (out of K_j, the number of clusters found in $\mathbf{B}_j$). After getting all block results and performing the aforementioned K-means, suppose $\hat{f}_{jk}(\boldsymbol{Z})$ is found to be in cluster $k' \in [K]$ (among the clusters found in $\{\hat{f}_{jk}(\boldsymbol{Z})\}$). Then, we assign $s_t = k'$ as the final output of Step 1. This simple method allows us to combine these block results very efficiently.

C.2 ITERATIVE UPDATES

In this step, we treat the problem as a HMM with Gaussian process emission with fixed K_T number of components, as obtained after combining the refined results from blocks. We iterate between the following, till convergence or based on a set maximum number of iterations: (1) Update s: Using current estimates of clusters and transition matrix, update $s_{1:T}$ using Viterbi algorithm, where the emission probabilities can be computed using Equation (15) in the main paper and (2) Update clusters: Using current $s_{1:T}$, for each k, collect $\mathcal{D}_{1:T}^{(k)}$ and fit a GP regression model, updating the kernel parameters. Thus, we end up with K GP models. In the case that $\mathcal{D}_{1:T}^{(k)} = \emptyset$, we reduce the number of clusters. This step is similar in spirit to K-Means. Fitting GP on the entire $\mathcal{D}_{1:T}^{(k)}$ is slow because of reasons described, hence we instead use SGP here as well. If the size of $\mathcal{D}_{1:T}^{(k)}$ is large, based on a relatively moderate subset of it, use GP to estimate the optimal kernel. Then, use this kernel along with the fixed inducing points and the entire data $\mathcal{D}_{1:T}^{(k)}$ to fit a SGP.

Remarks on run time: The run time of the algorithm depends on many things, primarily T, n (which control the total size of the data). Furthermore, it depends on the number of clusters K estimated, recall that the tth forward pass step requires K_t many inversions. Furthermore, many steps in the algorithm, including the last iterative steps, require computation of the emission probability and each time, we need to compute K many of them for each time t. Since SGP is used, the run time also depends on the size of the inducing points (since all matrix inversions are controlled by this size). Results in Table 2 in the main paper indicate that when the number of clusters is not large, iHMM-GP can analyze a total data of size around 75,000 in a couple of minutes.

D SIMULATION SETTINGS

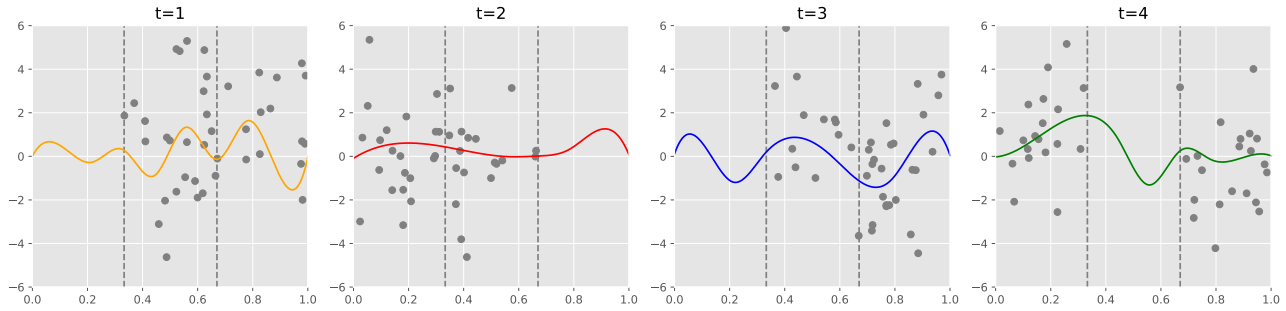


Figure 3: Examples of training data for $D = P = 1$ with censored locations (colored functions are the true clusters).

All required code to set up simulations and use the algorithm are provided at the author's Github repository.

For all experiments in Sections 5.3 and 5.4 in the main paper, we simulated the data for extra 100 time points, the observations corresponding to these last 100 time points were used as test data. Thus, the model was fit using the first T time points and then labels were predicted for these last 100 time points, which were also compared to the true labels to obtain the test scores. Furthermore, for these experiments we allowed Step 2 of the algorithm to run at most $L_{\max} = 5$ times.

D.1 CASE $D = P = 1$

Here we consider 4 functions, shown in Figure 3, which were generated by fitting a b-spline to 10 random observations in $\mathcal{X} = (0, 1)$. The pairwise distances between these functions are given in Fig 4.

We consider two types of situation under this case, both having $T = 400, \sigma^2 = 4$. In the first case (EXP1), we take $n = 60$ randomly chosen locations in $(0, 1)$ at each t. In the second case (EXP2), we censor a third of the data, i.e. use $n = 40$, split the spatial region into three equal parts and only take observations in two of them - at each time point, we select 2 of these 3 parts randomly and observe data there. This situation is shown in Figure 3. The true transition matrix Π is constructed as $\pi_{i,i} = 0.2$ and $\pi_{i,i+1} = 0.8$ for $i \in [4]$, where the last one is $\pi_{41} = 0.8$. This induces strong temporal dependence, which we wish to capture.

D.2 CASE $D = P = 2$

Here we discuss the setting for $D = P = 2$. In this case, we consider $K^* = 9$ true functions on $\mathcal{X} = (-1, 1) \times (-1, 1)$. We think of them into 3 types (every output dimension is constructed similarly within a type): for type 1, each is of the form $f(x) = a_0 \sin(b_0 x) + a_1 \cos(b_1 x) + a_2$, where all the coefficients were drawn randomly in $(-2, 2)$; for type 2, for each we fixed a set of 80 random values at 80 random locations and fit a basis spline. The functions in type 2 (4 in count) turned out to be more wiggly than those in type 1 (2 in count). Type 3 functions (the remaining 3) were created by spatially mixing two or more of these others. We give an example: consider the functions in Figure 5. The left one, indicated by f_4 in blue, is of type 1; the right one, indicated by f_1 in red is of type 2; while f_7 (in the middle, in purple) is constructed as $f_7(\boldsymbol{x}) = (1 - \epsilon(x_2))f_1(\boldsymbol{x}) + \epsilon(x_2)f_4(\boldsymbol{x})$, where $\boldsymbol{x} = (x_1, x_2)$ and $\epsilon(x) = 1/(1 + e^{-10x})$. Note that f_7 resembles f_1 in the lower part and resembles f_4 in the upper part. The true transition matrix is similar to the one for the univariate case.

For all simulations with these true parameters, we set $T = 600$. For the data, we perform a similar censoring (see Figure 2 in the main paper), where data in only a part of $\mathcal{X}$, not the whole, is observed at each time t. The L_∞ distances between pairs of them have a minimum of 1.2 and a maximum of 4.2, while the average L_2 distances range from 0.86 to 1.7. These are shown in Fig 6.

D.3 SIMULATIONS 1 CONTINUED

Here we present two other simulations, continuing from simulation Section in the main text.

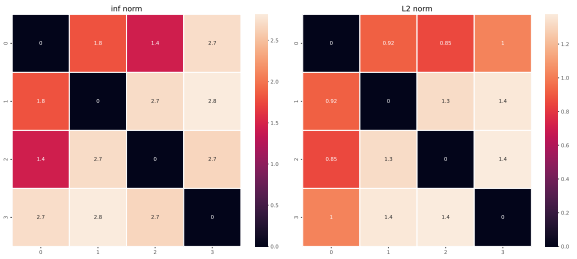


Figure 4: Distance between pairs of the 4 true functions for $D = P = 1$ case.

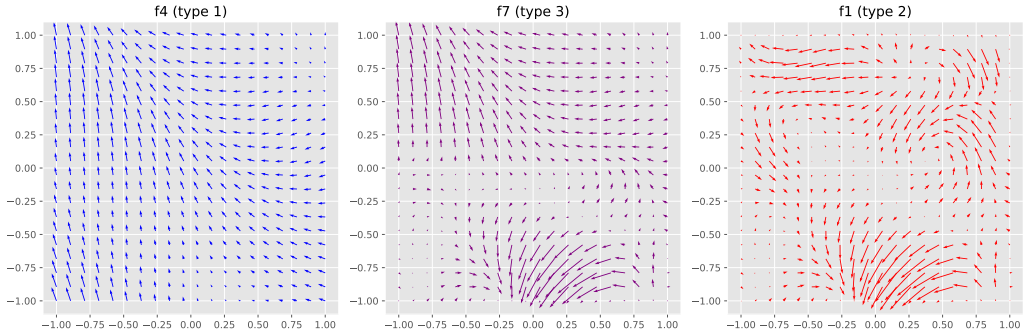


Figure 5: Quiver plots of 3 (out of 9) of the true functions used in simulation.

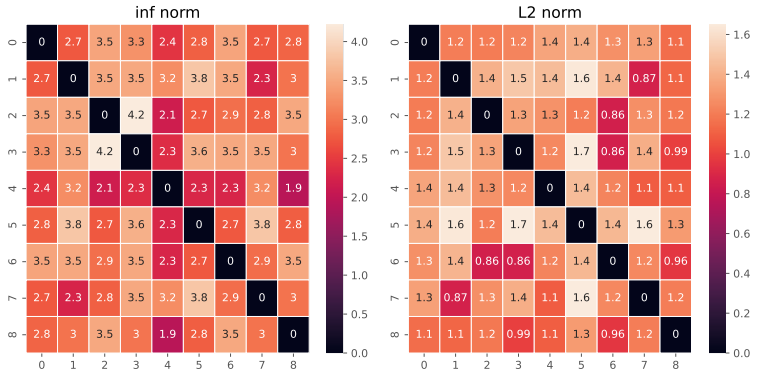
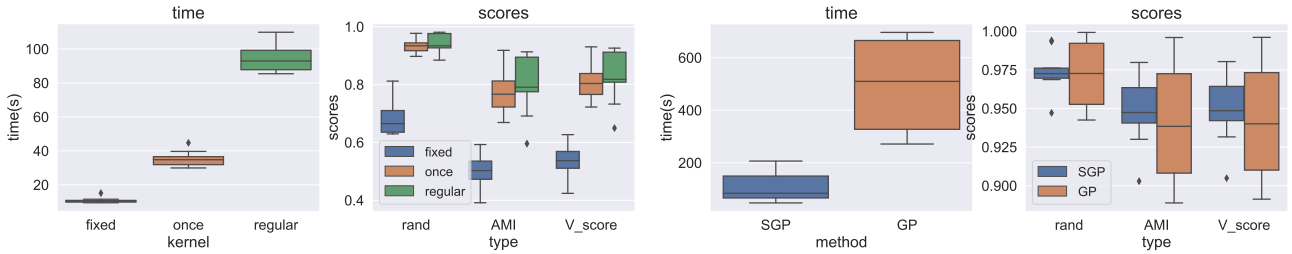


Figure 6: Distance between pairs of the 9 true functions for $D = P = 2$ case.

Impact of updating the kernel parameters: In our proposed algorithm, we estimate kernel parameters during initialization for each time t and use these throughout the entire forward pass. We compare the results of (a) fixed kernel with $\sigma_0 = 1, \ell_0 = 1$, (b) updating the kernel parameters only at the end of refinement and (c) updating after every 10 steps of the forward pass (denoted as *regular* in the figure). We take $T = 100, n = 50, \sigma^2 = 3$. The results are shown in Figure 7a. It seems using a fixed kernel reduces the performance significantly, small n is most likely the cause. However, we find that updating the kernel parameters regularly leads to slightly better performance than updating just once at the end. In terms of time, (a) took 14 sec on average, (b) took around 40 sec on average and (c) took 75 seconds on average.

GP versus SGP : We performed a simple experiment to compare the performance of the usual Gaussian process with the sparse Gaussian process. We take $T = 600, n = 30, \sigma^2 = 1$ for this experiment. For GP, we replace all the steps where we use sparse Gaussian process with the usual Gaussian process. The results are shown in Figure 7b. We see that the mean label estimation is similar for both methods, although, GP has a much higher variance. Also, we note that GP takes a much longer time to fit, which is one of the reasons why we use sparse GP in the first place. However, we note that even with GP, it takes around 500 seconds on average for this setup. In contrast, in Experiment 3, we noted that only 3 Gibbs iterations for DP-GP required 600 seconds - this shows that merely replacing GP with SGP does not solve the time complexity issue associated



(a) Effect of kernel updation.

(b) Comparison between using GP and SGP

Figure 7: Simulation 1 - effect of different kernel update strategies (left) and comparison between GP and SGP (right).

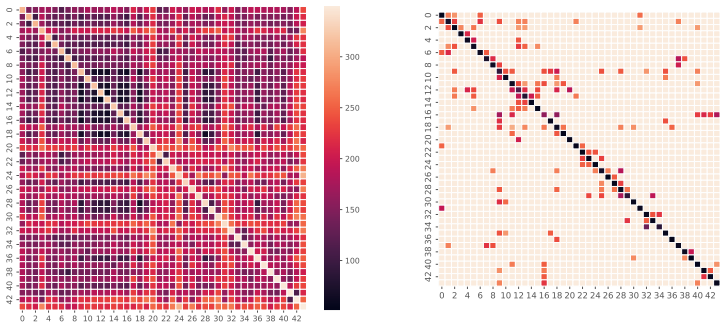


Figure 8: (left) Pairwise average L_2 distance between estimated velocity fields and (right) heatmap of estimated transition matrix.

with MCMC methods for such models.

D.4 OTHER SIMULATION DETAILS

Our proposed method, HMM with Gaussian emission and DP-GP were performed on a cluster with 32 cores, while HMM-GPSM was run on a local machine with 8 cores. G-HMM assumes Gaussian distribution emission, which does not exactly fit into our setup, we artificially convert the data to be used by it: for every t, we fit a GP and take the posterior mean over a selected fixed set of locations (size $w = 60$ for Exp 1 and 2 and size $w = 225$ for Exp 3), these $w-$dim vectors over T time points are used to fit the G-HMM model.

Since the other algorithms only returned the estimated state labels for the training data, we computed the log likelihood for all methods in the same way. Starting with the $\hat{s}_t$, we treat it like usual HMM. We fit GP to each cluster, from which we compute the emission probabilities $p(\boldsymbol{Y}_t|\boldsymbol{X}_t, s_t = k)$ for each k, t. We use the estimated states to compute the estimated transition matrix $\hat{\Pi}$ with $\pi_{ij} \propto \sum_{t\in[T-1]} \mathbf{1}(s_t = i, s_{t+1} = j)$. Using these, we invoke the forward message passing algorithm to compute the log likelihood. For each method, we compute this value, scaled by the number of time points and report the average of this value over repetitions for a particular experiment.

E FURTHER DETAILS ABOUT NGSIM DATA

The entire dataset contains detailed description of location of each vehicle at a time resolution of 0.1 seconds, we only work with the local coordinates; in terms of which the $y-$axis is along North-South direction, where higher y values mean further North. We extract only observations related to the location Lankershim Bld. The dataset contains many other information like vehicle ID, vehicle size, lane, intersection details, however we only use the location and speed. Using the location and the speed data, we construct the velocity vectors for each vehicle at each time point (direction computed based on different of position vectors at an interval of 0.1 s). We scaled the region inside a $(-1, 1) \times (-10, 10)$ box, maintaining the aspect ratio. The whole dataset contains observations from 8AM to 8:30AM, we only chose to work with the first 8 minutes.

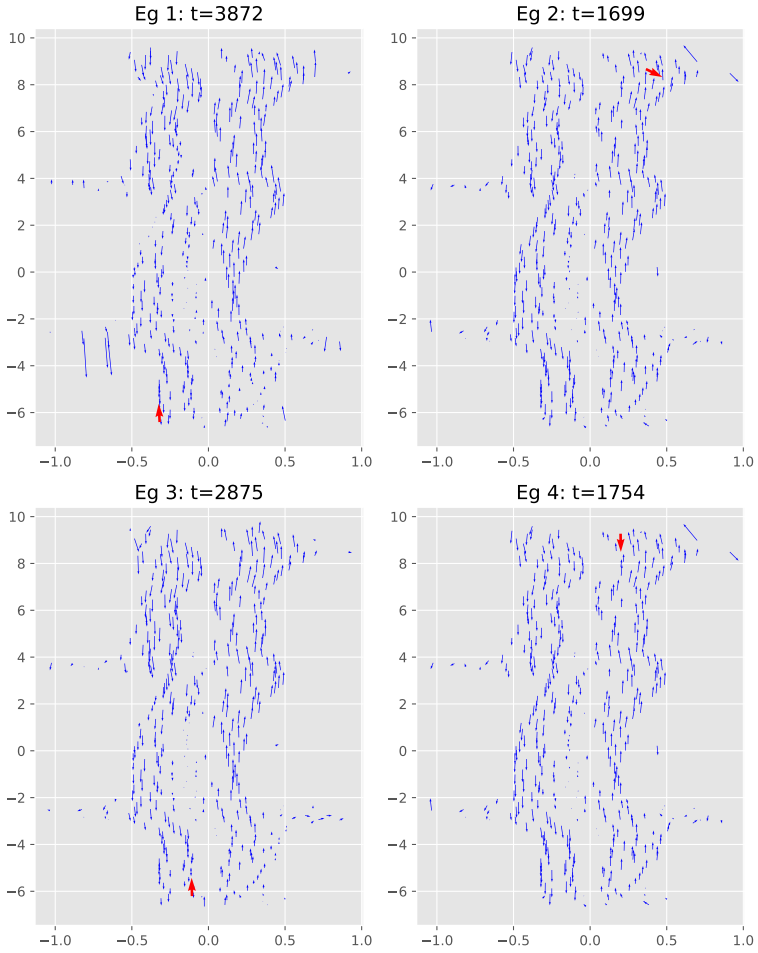


Figure 9: More examples of outliers.

We also notice the fact that there are regions where the estimated field is inconsistent with reality, e.g. if we look at the field evaluated near the MTA Dwy (top left part, around $(-0.6, 3.8)$), say in pattern 5, it shows a strong southwards velocity, which is absurd since vehicles are either EB or WB there. This is caused due to the spatial correlation induced by the GP (note that these are accompanied by very high downward velocity fields on the adjacent LB). On closely examining the time frames associated to these patterns, there were a very high number of such CB vehicles on this adjacent LB but few or no vehicles in the MTW Dwy.

We look into some of the other patterns in Fig 8 (in the main paper) in detail. Pattern 3 appears to be the steady movement of traffic on LB, however the field shows that NB vehicles heading towards the intersection with UHD are coming to a stop. This state transitions to patterns 6 (where there is cross traffic towards UHD), 2 (SB vehicles on LB taking left turn into UHD) or pattern 1 (WB vehicles emerging from UHD, going straight or turning right to LB). There were some patterns which were pretty close (low value of average L_2 distance), which showed to be dissimilar either at a specific small region or the velocity directions were similar (which resulted in similar looking quiver plots) but the average speed was different in some sub-regions. On closer inspection, some of thee patterns show some locally inconsistent behaviors, e.g. in patterns 1, 3 or 5, there is a strong motion towards south in the MTW Dwy, which is strange since vehicles only move towards east or west there. This is caused by the GPs, whose kernel induce strong spatial correlation to affect other locations - note that in these patterns, there is in fact, a strong SB velocity pattern adjacent to this area on the LB. Possibly using other information like lane or intersection ID or other road information might solve this problem, if we consider a spatial mixture of GP instead of a single one.

Outlier detection example: The goal was to determine any seemingly abnormal driving behavior of particular vehicles in the entire data. Using the estimated fields at each time, we predicted $\hat{\boldsymbol{Y}}_t = \hat{f}_{\hat{s}_t}(\boldsymbol{X})$ using the fitted model and computed

$\|y_{tj} - \hat{y}_{tj}\|$ for each $t \in [T], j \in [n_t]$. Looking at these prediction errors, we looked at the 10 highest points. Two such cases were shown in Figure 10 in the main paper. We show some more examples in Figure 9. The Eg 1 here is similar to the situation discussed in the paper, however this is at a very different time stamp (shown above the respective images). Eg 2 and 4 are in the similar region, most likely there is a left turn lane to move into Main St for SB vehicles on LB. Such left turn lanes are often adjacent to the left-most lane in the opposite direction (here for NB vehicles on LB). It is likely that based on all time points assigned to this pattern, there were many NB vehicles in that region and a handful in that left turn lane, which explains why the estimated traffic pattern is as shown. Again, using lane position information into such a study would improve the performance.