Appendices

A. Little & Rubin (1987)'s Missing-Data Taxonomy

Little & Rubin (1987)'s classical taxonomy of MNAR, MAR, and MCAR mechanisms[12] was meant for graphical models. A graphical model has a *fixed set* of random variables. The missingness mechanisms envisioned by Little & Rubin (1987) simply decide which of those variables are suppressed in a joint observation. For them, an observed sample always reveals *which* variables were observed, and thus it reveals *how many* variables are missing.

In contrast, our incomplete event stream is most simply described as a single random variable Y that is *partly* missing. If we tried to describe it using $|\mathbf{x} \sqcup \mathbf{z}|$ random variables with values like $k@t$, then the observed sample $\mathbf{x}$ would *not* reveal the number of missing variables $|\mathbf{z}|$ nor the total number of variables $|\mathbf{x} \sqcup \mathbf{z}|$. There would not be a fixed set of random variables.

To formulate our model in Little & Rubin's terms, we would need a fixed set of uncountably many random variables K_t where t ranges over the set of times. $K_t = k$ if there is an event of type k at time t, and otherwise $K_t = 0$. For some finite set of times t, we observe a specific value $K_t > 0$, corresponding to some observed event. For all other times t, the value of K_t is missing, meaning that we do not know whether or not there is any event at time t, let alone the type of such an event. A crucial point is that 0 values are never observed in our setting, because we are never told that an event did *not* happen at time t. In contrast, a value > 0 (corresponding to an event) may be either observed or unobserved. Thus, the probability that K_t is missing depends on whether $K_t > 0$, meaning that this setting is MNAR.

We preferred to present our model (section 2.1) in terms of the finite sequences that are generated or read by our LSTMs. This simplified the notation later in the paper. But it does not cure the MNAR property: see section 5.1.

Again, our presentation does not allow a Little & Rubin (1987) style formulation in terms of a finite fixed set of random variables, some of which have missing values. That formulation would work if we knew the total number of events I, and were simply missing the value k_i and/or t_i for some indices i. But in our setting, the number of events is itself missing: after each observed event i, we are missing J_i events where J_i is itself unknown. In other words, we need to impute even the number of variables in the complete sequence $\mathbf{x} \sqcup \mathbf{z}$, not just their values.

Our definition of MAR in section 2.1 is the correct generalization of Little & Rubin (1987)'s definition: namely, it is the case in which the second factor of equation (1) can be ignored. The ability to ignore that factor is precisely why anyone cares about the MAR case. This was mentioned at equation (1), and is discussed in conjunction with the EM algorithm in Appendix H.

Since missing-event settings tend to violate this desirable MAR property, all our experiments address MNAR problems. As Little & Rubin (1987) explained, the more general case of MNAR data cannot be treated without additional knowledge. The difficulty is that identifying p_{model} *jointly* with p_{miss} becomes impossible. If you observe few 50-year-olds on your survey, you cannot know (beyond your prior) whether that's because there are few 50-year-olds, or because 50-year-olds are very likely to omit their age.

Fortunately, we do have additional knowledge in our setting. Joint identification of p_{model} and p_{miss} is unnecessary if either (1) one has separate knowledge of the missingness distribution p_{miss}, or (2) one has separate knowledge of the complete-data distribution p_{model}. In fact, both (1) and (2) hold in the MNAR experiments of this paper (sections 5.1–5.2). But in general, if we know at least one of the distributions, then we can still infer the other (Appendix H).

A.1. Obtaining Complete Data

Readers might wonder why (2) above would hold in a missing-data situation. In practice, where would we obtain a dataset of complete event streams (as in section 5.2) for supervised training of $p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z})$?

In some event stream scenarios, a training dataset of complete event streams can be collected at extra expense. This is the hope in the medical and user-interface scenarios in section 1. For our imputation method to work on partially observed streams $\mathbf{x} \sqcup \mathbf{z}$, their complete streams should be distributed like the ones in the training dataset.

Other scenarios could be described as having **eventually complete** streams. Complete information about each event stream *eventually* arrives, at no extra expense, and that event stream can then be used for training. For example, in the competitive game scenario in section 1), perhaps including wars and political campaigns, each game's true complete history is revealed after the game is over and the need for secrecy has passed. While a game is underway, however, some events are still missing, and imputing them is valuable. Both (1) and (2) hold in these settings.

[12] **Missing not at random (MNAR)** makes no assumptions about the missingness mechanism. **Missing at random (MAR)** is a modeling assumption: determining from data whether the MAR property holds is "almost impossible" (Mohan & Pearl, 2018). **Missing completely at random (MCAR)** is a simple special case of MAR.

An interesting subclass of eventual completeness arises in monitoring settings such as epidemiology, journalism, and sensor networks. These often have **reporting delays**, so that one observes each event only some time after it happens. Yet one must make decisions at time $t < T$ based on the events that have been observed so far. This may involve imputing the past and predicting the future. The missingness mechanism for these reporting delays says that more recent events (soon before the current time t) are more likely to be missing. The probability that such an event would be missing depends on the specific distribution of delays, which can be learned with supervision once all the data have arrved.

We point out that in all these cases, the "complete" streams $\mathbf{x} \sqcup \mathbf{z}$ that are used to train p_{model} do not actually have to be *causally* complete. It may be that in the real world, there are additional latent events $\mathbf{w}$ that cause the events in $\mathbf{x} \sqcup \mathbf{z}$ or mediate their interactions. Mei & Eisner (2017, section 6.3) found that the neural Hawkes process was expressive enough in practice to ignore this causal structure and simply use $\mathbf{x} \sqcup \mathbf{z}$ streams to directly train a neural Hawkes process model $p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z})$ of the *marginal* distribution of $\mathbf{x} \sqcup \mathbf{z}$, without explicitly considering $\mathbf{w}$ in the model or attempting to sum over $\mathbf{w}$ values. The assumption here is the usual assumption that $\mathbf{x} \sqcup \mathbf{z}$ will have the same distribution in training and test data, and thus $\mathbf{w}$ will be missing in both, with the same missingness mechanism in both. By contrast, $\mathbf{z}$ is missing only in test data. It is not possible to impute $\mathbf{w}$ because it was not modeled explicitly, nor observed even in training data. However, it remains possible to impute $\mathbf{z}$ in test data based on its distribution in training data.

B. Complete Data Model Details

Our complete data model, such as a neural Hawkes process, gives the probability $p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z})$ that $\mathbf{x} \sqcup \mathbf{z}$ will be the complete set of events on a given interval $[0, T)$. this probability can always be written in the factored form

$$\Big(\prod_{i=0}^{I}\prod_{j=0}^{J_i} p(k_{i,j} @ t_{i,j} \mid \mathcal{H}(t_{i,j}))\Big) \cdot p(@ \geq T \mid \mathcal{H}(T)) \quad (14)$$

where $p(k @ t \mid \mathcal{H}(t))$ denotes the probability density that the first event following $\mathcal{H}(t)$ (which is the set of events occurring *strictly* before t) will be $k @ t$, and $p(@ \geq t' \mid \mathcal{H}(t))$ denotes the probability that this event will fall at some time $\geq t'$.

Thus, the final factor of (14) is the probability that there are no more events on $[0, T)$ following the last event of $\mathbf{x} \sqcup \mathbf{z}$. The initial factor $p(k_0 @ t_0 \mid \mathcal{H} t_0)$ is defined to be 1, since the boundary event $k_0 @ t_0$ is given (see section 2.1).

B.1. Neural Hawkes Process Details

In this section we elaborate slightly on section 2.2. Again, $\lambda_k(t \mid \mathcal{H}(t))$ is defined by equation (2) in terms of the hidden state of a *continuous-time* left-to-right LSTM. We spell out the continuous-time LSTM equations here; more details about them may be found in Mei & Eisner (2017).

$$\mathbf{h}(t) = \mathbf{o}_i \odot (2\sigma(2\mathbf{c}(t)) - 1) \text{ for } t \in (t_{i-1}, t_i] \quad (15)$$

where the interval $(t_{i-1}, t_i]$ has consecutive observations $k_{i-1} @ t_{i-1}$ and $k_i @ t_i$ as endpoints. At t_i, the continuous-time LSTM reads $k_i @ t_i$ and updates the current (decayed) hidden cells $\mathbf{c}(t)$ to new initial values $\mathbf{c}_{i+1}$, based on the current (decayed) hidden state $\mathbf{h}(t_i)$, as follows:[13]

$$\mathbf{i}_{i+1} \leftarrow \sigma\left(\mathbf{W}_{\mathrm{i}}\mathbf{k}_i + \mathbf{U}_{\mathrm{i}}\mathbf{h}(t_i) + \mathbf{d}_{\mathrm{i}}\right) \quad (16a)$$
$$\mathbf{f}_{i+1} \leftarrow \sigma\left(\mathbf{W}_{\mathrm{f}}\mathbf{k}_i + \mathbf{U}_{\mathrm{f}}\mathbf{h}(t_i) + \mathbf{d}_{\mathrm{f}}\right) \quad (16b)$$
$$\mathbf{z}_{i+1} \leftarrow 2\sigma\left(\mathbf{W}_{\mathrm{z}}\mathbf{k}_i + \mathbf{U}_{\mathrm{z}}\mathbf{h}(t_i) + \mathbf{d}_{\mathrm{z}}\right) - 1 \quad (16c)$$
$$\mathbf{o}_{i+1} \leftarrow \sigma\left(\mathbf{W}_{\mathrm{o}}\mathbf{k}_i + \mathbf{U}_{\mathrm{o}}\mathbf{h}(t_i) + \mathbf{d}_{\mathrm{o}}\right) \quad (16d)$$

$$\mathbf{c}_{i+1} \leftarrow \mathbf{f}_{i+1} \odot \mathbf{c}(t_i) + \mathbf{i}_{i+1} \odot \mathbf{z}_{i+1} \quad (17a)$$
$$\underline{\mathbf{c}}_{i+1} \leftarrow \underline{\mathbf{f}}_{i+1} \odot \underline{\mathbf{c}}_i + \underline{\mathbf{i}}_{i+1} \odot \mathbf{z}_{i+1} \quad (17b)$$
$$\boldsymbol{\delta}_{i+1} \leftarrow f\left(\mathbf{W}_{\mathrm{d}}\mathbf{k}_i + \mathbf{U}_{\mathrm{d}}\mathbf{h}(t_i) + \mathbf{d}_{\mathrm{d}}\right) \quad (17c)$$

The vector $\mathbf{k}_i \in \{0, 1\}^K$ is the i^{th} input: a one-hot encoding of the new event k_i, with non-zero value only at the entry indexed by k_i. Then, $\mathbf{c}(t)$ for $t \in (t_{i-1}, t_i]$ is given by (18), which continues to control $\mathbf{h}(t)$ except that i has now increased by 1).

$$\mathbf{c}(t) \stackrel{\text{def}}{=} \underline{\mathbf{c}}_{i+1} + \left(\mathbf{c}_{i+1} - \underline{\mathbf{c}}_{i+1}\right) \exp\left(-\boldsymbol{\delta}_{i+1}\left(t - t_i\right)\right) \quad (18)$$

On the interval $(t_i, t_{i+1}]$, $\mathbf{c}(t)$ follows an exponential curve that begins at $\mathbf{c}_{i+1}$ (in the sense that $\lim_{t \to t_i^+} \mathbf{c}(t) = \mathbf{c}_{i+1}$) and decays, as time t increases, toward $\underline{\mathbf{c}}_{i+1}$ (which it would approach as $t \to \infty$, if extrapolated).

The **intensity** $\lambda_k(t \mid \mathcal{H}(t)) \in \mathbb{R}_{\geq 0}$ may be thought of as the instantaneous *rate* of events of type k at time t. More precisely, as $dt \to 0^+$, the expected number of events of type k occurring in the interval $[t, t + dt)$, divided by dt, approaches $\lambda_k(t \mid \mathcal{H}(t))$. If no event of any type occurs in this interval (which becomes almost sure as $dt \to 0^+$), one may still occur in the next interval $[t + dt, t + 2dt)$, and so on. The intensity functions $\lambda_k(t \mid \mathcal{H}(t))$ are continuous on intervals during which no event occurs (note that $\mathcal{H}(t)$ is constant on such intervals). They jointly determine a distribution over the time of the next event after $\mathcal{H}(t)$, as used in every factor of equation (14). As it turns out (Mei

[13] The upright-font subscripts i, f, z and o are not variables, but constant labels that distinguish different $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{d}$ tensors. The $\underline{\mathbf{f}}$ and $\underline{\mathbf{i}}$ in equation (17b) are defined analogously to $\mathbf{f}$ and $\mathbf{i}$ but with different weights.

& Eisner, 2017), $\log p_{\text{model}}(Y = \mathbf{x} \sqcup \mathbf{z})$ becomes

$$\sum_{\ell} \log \lambda_{k_\ell}(t_\ell \mid \mathcal{H}(t_\ell)) - \int_{t=0}^{T} \sum_{k=1}^{K} \lambda_k(t \mid \mathcal{H}(t)) dt \quad (19)$$

where the first sum ranges over all events $k_\ell @ t_\ell$ in $\mathbf{x} \sqcup \mathbf{z}$.

We can therefore train the parameters $\boldsymbol{\theta}$ of the λ_k functions by maximizing log-likelihood on training data. The first term of equation (19) can be differentiated by backpropagation. Mei & Eisner (2017) explain how simple Monte Carlo integration (see also our Appendix C.3) gives an unbiased estimate of the second term of equation (19), and how the random terms in the Monte Carlo estimate can similarly be differentiated to give a stochastic gradient.

C. Sequential Monte Carlo Details

Our main algorithm is presented as Algorithm 1. It covers both particle filtering and particle smoothing, with optional multinomial resampling.

In this section, we provide some additional details and notes on the design and operation of the pseudocode.

C.1. Explicit Formula for the Proposal Distribution

The proposal distribution $q(\mathbf{z} \mid \mathbf{x})$ factors as follows, and the pseudocode uses this factorization to construct $\mathbf{z}$ by sampling its individual events from left to right:

$$\prod_{i=0}^{I} \Big(\prod_{j=1}^{J_i} \big(q(k_{i,j} @ t_{i,j} \mid \mathcal{H}(t_{i,j}), \mathcal{F}(t_{i,j})) \big) \cdot q(@ \geq t_{i+1} \mid \mathcal{H}(t_{i+1}), \mathcal{F}(t_{i,J_i})) \Big) \quad (20)$$

Here the notation for $q(\cdot \mid \cdot)$ is the same as that for $p(\cdot \mid \cdot)$ in Appendix B. However, the $q(\cdot \mid \cdot)$ terms are proposal probabilities that condition on different evidence—not only the set $\mathcal{H}(t)$ of all events (observed and unobserved) at times $< t$, but also the set $\mathcal{F}(t)$ of events at times $> t$.[14] All of the proposal probabilities $q(\cdot \mid \cdot)$ are determined by the intensity functions in (5).

We can sample $\mathbf{z}$ from $q(\mathbf{z} \mid \mathbf{x})$ in chronological order: for each $0 \leq i \leq I$ in turn, draw a sequence of J_i unobserved events that follow the observed event $k_i @ t_i$. The probabilities of these J_i events are the inner factors in equation (20). This sequence ends (thereby determining J_i) if the next proposed event would have fallen after t_{i+1} and thus is preempted by the observed event $k_{i+1} @ t_{i+1}$. The probability of so ending the sequence corresponds to the $q(@ \geq t_{i+1} \mid \cdots)$ factor in equation (20).

[14] In particular, the second q factor above is the probability that the event at time t_{i,J_i} is the last one before t_{i+1}, given knowledge of all past events up through and including the one at t_{i,J_i}, and all future observed events starting with the one at t_{i+1}.

Equation (20) resembles equation (14), but it conditions each proposed unobserved event not only on the history but also on the future. Section 3.2.1 tries to train $q(\mathbf{z} \mid \mathbf{x})$ to approximate the target distribution $p(\mathbf{z} \mid \mathbf{x})$, by making $q(\cdot \mid \mathcal{H}, \mathcal{F}) \approx p(\cdot \mid \mathcal{H}, \mathcal{F})$. In other words, at each step, q should draw the next proposed event approximately from the posterior of the model p, even though we have no closed form computation for that posterior.

Just as equation (14) yields the formula (19) for $\log p_{\text{model}}$ when we use a neural Hawkes process model, equation (20) yields the following formula for $\log q(\mathbf{z} \mid \mathbf{x})$ when we use the proposal intensities from (5):

$$\sum_{\ell} \log \lambda^q_{k_\ell}(t_\ell \mid \mathcal{H}(t_\ell), \mathcal{F}(t_\ell)) - \int_{t=0}^{T} \sum_{k=1}^{K} \lambda^q_k(t \mid \mathcal{H}(t), \mathcal{F}(t))\, dt \quad (21)$$

where the first sum ranges over all events $k_\ell @ t_\ell$ in $\mathbf{z}$ only.

C.2. Managing LSTM State Information

The push and pop operations shown in the pseudocode must be implemented so that they also have the effect of updating LSTM configurations.

Our p_{model} uses a left-to-right LSTM to construct its state after reading all events so far from left to right (section 2.2). Since each particle posits a different event sequence, we maintain a separate LSTM configuration for each particle $m = 1, 2, \ldots, M$. If *smooth* = **true**, our q additionally uses a right-to-left LSTM whose state has read all future observed events from right to left (section 3.2). We maintain the configuration of this LSTM as well.

Specifically, in Algorithm 1, when we push an event to the stack $\mathcal{H}_m$ (lines 3 and 44), we update the configuration of particle m's left-to-right LSTM (including gates, cell memories and states).

If *smooth* = **true**, then when we push an event to the stack $\mathcal{F}$ (line 6), we update the configuration of the right-to-left LSTM. Moreover, before updating that configuration, we push it onto an parallel stack, so that we can revert the update when we later *pop* the event from $\mathcal{F}$ (line 34).

These LSTM configurations provide the $\mathbf{h}(t)$ and $\bar{\mathbf{h}}(t)$ vectors for the computation of intensities $\lambda^p_k(t)$ and $\lambda^q_k(t)$ in equations (2) and (5). These intensities are needed in lines 41 and 43 of the algorithm.

C.3. Integral Computation

Mei & Eisner (2017, section B.2) construct a Monte Carlo estimator of the $\int_0^T$ integral in equation (19), by evaluating $\sum_k \lambda_k(t \mid \mathcal{H}(t)) \cdot T$ at a random $t \sim \text{Unif}(0, 1)$. While even one such sample would provide an unbiased estimate,

Algorithm 1 Sequential Monte Carlo — Neural Hawkes Particle Filtering/Smoothing

Input: observed sequence $\mathbf{x} = k_0@t_0, \ldots, k_{I+1}@t_{I+1}$ with $t_0 = 0, t_{I+1} = T$;
model p; missingness mechanism p_{miss}; proposal distribution q; number of particles M;
boolean flags *smooth* and *resample*

Output: collection $\{(\mathbf{z}_1, w_1), \ldots, (\mathbf{z}_M, w_M)\}$ of weighted particles

procedure SEQUENTIALMONTECARLO($\mathbf{x}, p, p_{\text{miss}}, q, M, smooth, resample$)
  for $m = 1$ **to** M : ▷ *init weighted particles* $(\mathbf{z}_m, w_m)$. *History* $\mathcal{H}_m$ *combines* $\mathbf{z}_m$ *with a prefix of* $\mathbf{x}$
    $\mathbf{z}_m \leftarrow$ empty sequence; $w_m \leftarrow 1$; $\mathcal{H}_m \leftarrow$ empty stack; push $k_0@t_0$ onto $\mathcal{H}_m$
  $\mathcal{F} \leftarrow$ empty stack
  for $i = I$ **downto** 0 : ▷ *stack of all future observed events*
    push $k_{i+1}@t_{i+1}$ onto $\mathcal{F}$ ▷ *as we reach these events, we'll pop from* $\mathcal{F}$ *and push onto* $\mathcal{H}_m$ $(\forall m)$
  for $i = 0$ **to** I : ▷ *propose unobserved events on interval* (t_i, t_{i+1}), *then observe next event* $k_{i+1}@t_{i+1}$
    for $m = 1$ **to** M :
      DRAWSEGMENT(i, m) ▷ *destructively extend* $\mathbf{z}_m, w_m, \mathcal{H}_m$ *with events on* $(t_i, t_{i+1}]$
    if *resample* & LOWESS() : RESAMPLE() ▷ *optional multinomial resampling replaces all weighted particles*
  return $\{(\mathbf{z}_m, w_m / \sum_{m=1}^M w_m)\}_{m=1}^M$ ▷ M *particles with weights normalized as in equation* (3)
procedure LOWESS ▷ *check if effective sample size is low*
  ESS $\leftarrow (\sum_{m=1}^M w_m)^2 / \sum_{m=1}^M (w_m)^2$
  if ESS $< M/2$: **return true**
  return false
procedure RESAMPLE ▷ *has access to global variables*
  for $m = 1$ **to** M : ▷ *often draws multiple copies of good (high-weight) particles, 0 copies of bad ones*
    $\tilde{\mathbf{z}}_m \sim \text{Categorical}(\{\mathbf{z}_m \mapsto \frac{w_m}{\sum_{m=1}^M w_m}\}_{m=1}^M)$
  for $m = 1$ **to** M :
    $\mathbf{z}_m \leftarrow \tilde{\mathbf{z}}_m$; $w_m \leftarrow 1$ ▷ *update particles and their weights*
procedure DRAWSEGMENT(i, m) ▷ *has access to global variables*
  ▷ p *gives info to define function* $\lambda_k^p(t) \stackrel{\text{def}}{=} \lambda_k(t \mid \mathcal{H}_m)$
  ▷ q *gives info to define function* $\lambda_k^q(t) \stackrel{\text{def}}{=} \lambda_k(t \mid \mathcal{H}_m, \mathcal{F})$, *or simply* $\lambda_k^q(t) = \lambda_k^p(t)$ *if* $smooth =$ ***false***
  ▷ *these functions consult the state of a left-to-right LSTM that's read* $\mathcal{H}_m$ *and possibly a right-to-left LSTM that's read* $\mathcal{F}$
  ▷ *we also define the* ***total intensity functions*** $\lambda^p(t) \stackrel{\text{def}}{=} \sum_{k=1}^K \lambda_k^p(t)$ *and* $\lambda^q(t) \stackrel{\text{def}}{=} \sum_{k=1}^K \lambda_k^q(t)$
  $i' \leftarrow i$; $j \leftarrow 0$; $t \leftarrow t_i$ ▷ *where* t_i *is time of top element of* $\mathcal{H}_m$
  repeat ▷ *each iteration adds a new event with index* $\langle i', j \rangle = \langle i, 1 \rangle, \ldots, \langle i, J_i \rangle, \langle i+1, 0 \rangle$
    $j \leftarrow j + 1$
    repeat ▷ *thinning algorithm (see Mei & Eisner, 2017)*
      find any $\lambda^* \geq \sup \{\lambda^q(t') : t' \in (t, t_{i+1}]\}$ ▷ *e.g., old* λ^* *still works if* i *unchanged; see Appendix C.4*
      draw $\Delta \sim \text{Exp}(\lambda^*)$, $u \sim \text{Unif}(0, 1)$
      $t \mathrel{+}= \Delta$ ▷ *time of next proposed event (before thinning)*
      if $t > t_{i+1}$: ▷ *where* t_{i+1} *is time of top element of* $\mathcal{F}$
        $k@t \leftarrow$ pop $\mathcal{F}$; $i' \leftarrow i + 1$; $j \leftarrow 0$; ▷ *preempt proposed event by* $k_{i+1}@t_{i+1}$ *(popped from future into present)*
        break
    until $u\lambda^* \leq \lambda^q(t)$ ▷ *thinning: accept proposed time* t *only with prob* $\frac{\lambda^q(t)}{\lambda^*} \leq 1$
    ▷ *we've now chosen next event time* $t_{i',j}$ *to be* t; *let* t_{prev} *denote time of top element of* $\mathcal{H}_m$
    if $i' = i$: ▷ *it's a missing event*
      draw $k \in \{1, \ldots, K\}$ where probability of k is proportional to $\lambda_k^q(t)$ ▷ *choose next event type* $k_{i',j}$
      append $k@t$ to $\mathbf{z}_m$
      $w_m \leftarrow w_m / \exp\left(-\int_{t'=t_{\text{prev}}}^t \lambda^q(t')\, dt'\right) \cdot \lambda_k^q(t)$ ▷ *new factor within* q *in denominator of* (3); *see Appendix C.3*
    if $i' \leq I$: ▷ *skip final boundary event* $I + 1$ *(not generated by* p_{model}*)*
      $w_m \leftarrow w_m \cdot \exp\left(-\int_{t'=t_{\text{prev}}}^t \lambda^p(t')\, dt'\right) \cdot \lambda_k^p(t)$ ▷ *new factor within* p_{model} *in numerator of* (3); *see Appendix C.3*
      push $k@t$ onto $\mathcal{H}_m$ ▷ *event* $\langle i, j \rangle$ *just generated now becomes part of the past*
      $w_m \leftarrow w_m \cdot p_{\text{miss}}((k@t \in Z) = (i' = i) \mid \mathcal{H}_m)$ ▷ *new factor within* p_{miss} *in numerator of* (3): *missing or obs*
    until $i' = i + 1$

they draw $N = O(I)$ such samples, where I is the number of events, and average over these samples. This reduces the variance of the estimator, which decreases as $O(1/N)$. Notice that because they sample the N time points uniformly on $[0, T)$, longer intervals between observed events will tend to contain more points, which is appropriate.

Mei & Eisner (2017) (Appendix C.2) found that rather few samples could be used to estimate the integral. Indeed, even sampling at only I time points gave a standard deviation of log-likelihood—for the whole sequence—that was on the order of 0.1% of absolute (Mei, p.c.).

Our particle methods involve *comparing probabilities*. For each observed sequence $\mathbf{x}$, we use (3) to reweight the M particles according to their probability under the model divided by their probability under the proposal distribution. This means contrasting *two* probabilities for each particle (the p and q probabilities). It also means *comparing* the resulting probability ratios across all M particles, resulting in the normalized weights of equation (3).

For each of the M particles, the p_{model} factor in equation (3) is obtained by exponentiating equation (19), while the q factor is obtained by exponentiating equation (21). This means that each of these $2M$ factors contains the exp of an integral. To make all of these integral estimates more comparable and thus reduce the variance in the importance weights w_m (equation (3)), we evaluate all $2M$ integrals at the same set of N time points (see Appendix G.8). This practice ensures a "paired comparison" among particles: w_m and $w_{m'}$ differ only because they have different intensities at the sampled points, and not also because they sample at different points.

In Algorithm 1, these integral estimates are accumulated gradually at lines 41 and 43. The idea is that particle $\mathbf{z}_m$ partitions $[0, T)$ into the intervals between successive events of $\mathbf{x} \sqcup \mathbf{z}_m$. Thus, the (estimated) integral over $[0, T)$ can be expressed by summing the (estimated) integrals over these intervals. The estimate over an interval uses only the small subset of the N time points that fall into the interval. When we exponentiate the integrals to convert from log-probabilities into probabilities, this sum turns into a product, as shown at lines 41 and 43.

This gradual accumulation method gives the same result as if we had computed each integral "all at once" before exponentiating. However, it is useful to begin weighting the particles before they are complete. After each event $k_i @ t_i$ (for $0 \le i \le I + 1$), the partial particles up through this event already have partial weights w_m. It is these partial weights that are used by the RESAMPLE procedure (when $\mathit{resample} = \textbf{true}$).

In all experiments in this paper, we first sampled $I + 1$ points uniformly on $[0, T)$, for an average of only 1 time point per interval. In addition, for each interval (t_i, t_{i+1}), we sampled 1 point uniformly on that interval if it did not yet contain any points. Thus, $N \in [I + 1, 2I + 1]$.

Sampling at more points might be wise in settings where there are many missing events per interval (e.g., large ρ in section 5.1). This is especially true when $\mathit{resample} = \textbf{true}$. Resampling allows us to try multiple extensions of a high-weight particle; at the next resampling step, we prefer to keep the extensions that fared best. The danger is that if only a few sampling points happen to fall between resampling steps, then we may make a poor (high-variance) estimate of which extensions fared best.

For our setting, however, we found only negligible changes in the results by increasing to 5 time points per interval (i.e., sampling $5I+5$ points at the first step). Our evaluation metric (the minimum of (13) over all alignments $\mathbf{a}$) became slightly better for some values of C and slightly worse for others, but never by more than 2% relative. This is about the same variance that we get across different runs (with different random seeds) that have 1 time point per interval. Thus, we report only the results of the faster scheme. We caution that this hyperparameter might be more important in other settings.

C.4. Choice of λ^*

How do we construct the upper bound λ^* (line 30 of Algorithm 1)? For particle filtering, we follow the recipe in B.3 of Mei & Eisner (2017): we can express $\lambda^* = f_k(\max_t g_1(t) + \ldots + \max_t g_n(t))$ where each summand $v_{kd} h_d(t) = v_{kd} \cdot o_{id} \cdot (2\sigma(2c_d(t)) - 1)$ is upper-bounded by $\max_{c \in \{c_{id}, \bar{c}_{id}\}} v_{kd} \cdot o_{id} \cdot (2\sigma(2c) - 1)$. Note that the coefficients v_{kd} may be either positive or negative.

For particle smoothing, we simply have more summands inside f_k so $\lambda^* = f_k(\max_t g_1(t) + \ldots + \max_t g_n(t) + \max_t \bar{g}_1(t) + \ldots + \max_t \bar{g}_{\bar{n}}(t))$ where each extra summand $u_{kd} \bar{h}_d(t) = u_{kd} \cdot \bar{o}_{id} \cdot (2\sigma(2\bar{c}_d(t)) - 1)$ is upper-bounded by $\max_{c \in \{\bar{c}_{id}, \bar{\bar{c}}_{id}\}} u_{kd} \cdot \bar{o}_{id} \cdot (2\sigma(2c) - 1)$ and each u_{kd} is the d-th element of vector $\mathbf{v}_k^\top \mathbf{B}$ (equation (5)). Note that the $\bar{o}_{id}, \bar{c}_{id}, \bar{\bar{c}}_{id}$ of newly added summands $\bar{g}$ are actually from the right-to-left LSTM while those of g are from the left-to-right LSTM.

C.5. Missing Data Factors in p

Recall that the joint model (1) includes a factor $p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z})$, which appears in the numerator of the unnormalized importance weight (3). Regardless of the form of this factor, it could be multiplied into the particle's weight $\tilde{w}_m$ at the *end* of sampling (line 11 of Algorithm 1).

However, for some p_{miss} distributions, there is a better way. Algorithm 1 assumes that the missingness of each event

$k@t$ depends only on that event and preceding events,[15] so that $p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z})$ factors as

$$\prod_{\ell \in \text{indices}(\mathbf{z})} p_{\text{miss}}(k_\ell @ t_\ell \in Z \mid \{k_{\ell'} @ t_{\ell'} : \ell' \leq \ell\}) \cdot \prod_{\ell \in \text{indices}(\mathbf{x})} p_{\text{miss}}(k_\ell @ t_\ell \notin Z \mid \{k_{\ell'} @ t_{\ell'} : \ell' \leq \ell\}) \tag{22}$$

Algorithm 1 can thus *incrementally* incorporate the subfactors of equation (22), and does so at line 45 of Algorithm 1. For example, with the missingness mechanism in our experiments, equation (12), the p_{miss} factor in line 45 is ρ_k if the event is unobserved (that is, $i' = i$) or $1 - \rho_k$ if it is observed.

These subfactors are therefore taken into account as the particles are constructed, and thus play a role in resampling.

C.6. Optional Missing Data Factors in q

We can optionally improve the particle filtering proposal intensities to incorporate the p_{miss} factor discussed in Appendix C.5 (in which case that factor will be multiplied into the denominator of (3) and not just the numerator). This makes $q(\mathbf{z} \mid \mathbf{x})$ better match $p(\mathbf{z} \mid \mathbf{x})$: it means we will rarely posit an unobserved event that would rarely have gone missing.

Specifically, if a completed-data event $k@t$ would have probability $\rho_k(t \mid \mathcal{H}(t))$ of going missing given the preceding events $\mathcal{H}(t)$, it is wise to define $\lambda_k^q(t \mid \mathcal{H}(t)) = \lambda_k^p(t \mid \mathcal{H}(t)) \cdot \rho_k(t \mid \mathcal{H}(t))$.

We do include this extra ρ_k factor when defining λ_k^q for our experiments (section 5); that is, we modify the definition of λ_k^q at line 23. The factor is particularly simple in our experiments, where ρ_k is constant for each k.

Was this factor really necessary in the case of particle smoothing? One might say no: particle smoothing already tries to ensure through training that the proposal distribution will incorporate p_{miss}. That is because section 3.2.1 aims to train $\lambda_k^q(t \mid \mathcal{H}(t), \mathcal{F}(t))$ so that the resulting $q(\mathbf{z} \mid \mathbf{x}) \approx p(\mathbf{z} \mid \mathbf{x})$, and the posterior distribution $p(\mathbf{z} \mid \mathbf{x})$ does condition on the missingness of $\mathbf{z}$.

Still, if the ρ_k factor is known, why not include it explicitly in the proposal distribution, instead of having to train the BiLSTM to mimic it? Thus, in effect, we have modified the right-hand side of equation (5) to include a factor of ρ_k. This yields a more expressive and better-factored family of proposal distributions: missingness is now handled by the known ρ_k factor and the BiLSTM does not have to explain it. Additionally, our proposal distribution becomes more conservative about proposing missing events, because having a lot of missing events is *a posteriori* improbable. In other words, p_{miss} as given in equation (12) falls off with the number of missing events $|\mathbf{z}|$.

Modifying equation (5) in this way is particularly useful in the special case $r_k = 0$ (i.e., event type k is never missing and should not be proposed). There, it enforces the hard constraint that $\lambda_k^q = 0$ (something that the BiLSTM by itself could not achieve); and since this constraint is enforced regardless of the BiLSTM parameters, the events of type k appropriately become irrelevant to the training of the BiLSTM, which can focus on predicting other event types.

C.7. Events with Equal Times

In contrast to the notation in the main paper, our pseudocode is written in terms of sequence of events, rather than sets of events. As a result, it can handle the generalization noted in footnote 4, where a 0 delay is allowed between an event and the preceding event in the complete sequence. If this occurs, it means that multiple events have fallen at the same time—yet they still have a well-defined order in which they are generated and read by the LSTM.

An unobserved event may have a 0 delay, if line 31 proposes $\Delta = 0$ and the proposal is accepted. The neural Hawkes model can make in principle make such a proposal, but it has zero probability. However, it might have positive probability under a slightly different model.

An observed event may also have a 0 delay, if $t = t_{i+1}$ at line 33 and the proposal is accepted.[16] In this way, it is possible for the proposal distribution to propose any number of unobserved events at time t_{i+1} and immediately before the actual observed event $k_{i+1}@t_{i+1}$. However, once the proposal distribution happens to propose $\Delta > 0$, the actual observed event $k_{i+1}@t_{i+1}$ will preempt the proposal, ending this sequence of J_i unobserved events.

D. Right-to-Left Continuous-Time LSTM

Here we give details of the right-to-left LSTM from section 3.2. Note that this set of formulas is nearly the same as that of Appendix B.1—after all, it is a continuous-time LSTM that has the same architecture but different parameters from the one in the neural Hawkes process. The difference is that it reads only the observed events, and does so from right to left.

At each time $t \in (0, T)$, its hidden state $\bar{\mathbf{h}}(t)$ is continually

[15] This assumption could trivially be relaxed to allow it to also depend on the missingness of the preceding events, and/or on the future observed events $\mathcal{F}(t)$.

[16] It may seem improbable to propose $t = t_{i+1}$ exactly, but if $t_i = t_{i+1}$, then proposing an unobserved event between these two observed events is just a case of proposing with 0 delay, as in the previous paragraph.

obtained from the memory cells $\mathbf{c}(t)$ as the cells decay:

$$\bar{\mathbf{h}}(t) = \mathbf{o}_i \odot (2\sigma(2\bar{\mathbf{c}}(t)) - 1) \text{ for } t \in (t_{i-1}, t_i] \quad (23)$$

where the interval (t_{i-1}, t_i) has consecutive observations $k_{i-1}@t_{i-1}$ and $k_i@t_i$ as endpoints. At t_i, the continuous-time LSTM reads $k_i@t_i$ and updates the current (decayed) hidden cells $\bar{\mathbf{c}}(t)$ to new initial values $\bar{\mathbf{c}}_{i-1}$, based on the current (decayed) hidden state $\bar{\mathbf{h}}(t_i)$, as follows:

$$\bar{\mathbf{i}}_{i-1} \leftarrow \sigma\left(\mathbf{W}_{\mathrm{i}}\mathbf{k}_i + \mathbf{U}_{\mathrm{i}}\bar{\mathbf{h}}(t_i) + \mathbf{d}_{\mathrm{i}}\right) \quad (24a)$$
$$\bar{\mathbf{f}}_{i-1} \leftarrow \sigma\left(\mathbf{W}_{\mathrm{f}}\mathbf{k}_i + \mathbf{U}_{\mathrm{f}}\bar{\mathbf{h}}(t_i) + \mathbf{d}_{\mathrm{f}}\right) \quad (24b)$$
$$\bar{\mathbf{z}}_{i-1} \leftarrow 2\sigma\left(\mathbf{W}_{\mathrm{z}}\mathbf{k}_i + \mathbf{U}_{\mathrm{z}}\bar{\mathbf{h}}(t_i) + \mathbf{d}_{\mathrm{z}}\right) - 1 \quad (24c)$$
$$\bar{\mathbf{o}}_{i-1} \leftarrow \sigma\left(\mathbf{W}_{\mathrm{o}}\mathbf{k}_i + \mathbf{U}_{\mathrm{o}}\bar{\mathbf{h}}(t_i) + \mathbf{d}_{\mathrm{o}}\right) \quad (24d)$$

$$\bar{\mathbf{c}}_{i-1} \leftarrow \bar{\mathbf{f}}_{i-1} \odot \bar{\mathbf{c}}(t_i) + \bar{\mathbf{i}}_{i-1} \odot \bar{\mathbf{z}}_{i-1} \quad (25a)$$
$$\underline{\bar{\mathbf{c}}}_{i-1} \leftarrow \underline{\bar{\mathbf{f}}}_{i-1} \odot \underline{\bar{\mathbf{c}}}_i + \underline{\bar{\mathbf{i}}}_{i-1} \odot \bar{\mathbf{z}}_{i-1} \quad (25b)$$
$$\bar{\boldsymbol{\delta}}_{i-1} \leftarrow f\left(\mathbf{W}_{\mathrm{d}}\mathbf{k}_i + \mathbf{U}_{\mathrm{d}}\bar{\mathbf{h}}(t_i) + \mathbf{d}_{\mathrm{d}}\right) \quad (25c)$$

The vector $\mathbf{k}_i \in \{0, 1\}^K$ is the i^{th} input: a one-hot encoding of the new event k_i, with non-zero value only at the entry indexed by k_i. Then, $\bar{\mathbf{c}}(t)$ for $t \in (t_{i-1}, t_i]$ is given by (26), which continues to control $\bar{\mathbf{h}}(t)$ except that i has now decreased by 1).

$$\bar{\mathbf{c}}(t) \stackrel{\text{def}}{=} \underline{\bar{\mathbf{c}}}_{i-1} + \left(\bar{\mathbf{c}}_{i-1} - \underline{\bar{\mathbf{c}}}_{i-1}\right) \exp\left(-\bar{\boldsymbol{\delta}}_{i-1}\left(t_i - t\right)\right) \quad (26)$$

On the interval $[t_{i-1}, t_i)$, $\bar{\mathbf{c}}(t)$ follows an exponential curve that begins at $\bar{\mathbf{c}}_{i-1}$ (in the sense that $\lim_{t \to t_i^-} \bar{\mathbf{c}}(t) = \bar{\mathbf{c}}_{i-1}$) and decays, as time t decreases, toward $\underline{\bar{\mathbf{c}}}_{i-1}$.

E. Optimal Transport Distance Details

Pseudocode is presented in Algorithm 2 for finding optimal transport distance and the corresponding alignment. In the remainder of this section, we prove that optimal transport distance is a valid metric.

It is trivial that OTD is non-negative, since movement, deletion and insertion costs are all positive.

It is also trivial to prove that the following statement is true:

$$L(\mathbf{z}_1, \mathbf{z}_2) = 0 \Leftrightarrow \mathbf{z}_1 = \mathbf{z}_2, \quad (27)$$

where $\mathbf{z}_1$ and $\mathbf{z}_2$ are two sequences. If $\mathbf{z}_1$ is not identical to $\mathbf{z}_2$, the distance of them must be larger than 0 since we have to do some movement, insertion or deletion to make them exactly matched, so the right direction of equation (27) holds. If the distance between $\mathbf{z}_1$ and $\mathbf{z}_2$ is zero, which means they are already matched without any operations, $\mathbf{z}_1$ and $\mathbf{z}_2$ must be identical, thus the left direction of equation (27) holds.

OTD is symmetric, that is, $L(\mathbf{z}_1, \mathbf{z}_2) = L(\mathbf{z}_2, \mathbf{z}_1)$, if we set $C_{\text{insert}} = C_{\text{delete}}$. Suppose that $\mathbf{a}$ is an alignment between $\mathbf{z}_1$ and $\mathbf{z}_2$. It's easy to see that the only difference between $D(\mathbf{z}_1, \mathbf{z}_2, \mathbf{a})$ and $D(\mathbf{z}_2, \mathbf{z}_1, \mathbf{a})$ [17] is that the insertion and deletion operations are exchanged. For example, if we delete a token $t_i \in \mathbf{z}_1$ when calculating $D(\mathbf{z}_1, \mathbf{z}_2, \mathbf{a})$, we should insert a token at t_i to $\mathbf{z}_2$ when calculating $D(\mathbf{z}_2, \mathbf{z}_1, \mathbf{a})$. If we set $C_{\text{insert}} = C_{\text{delete}}$, we have

$$D(\mathbf{z}_1, \mathbf{z}_2, \mathbf{a}) = D(\mathbf{z}_2, \mathbf{z}_1, \mathbf{a}), \quad \forall \mathbf{a} \in \mathcal{A}(\mathbf{z}_1, \mathbf{z}_2). \quad (28)$$

Therefore, we could obtain

$$\begin{aligned} L(\mathbf{z}_1, \mathbf{z}_2) &= \min_{\mathbf{a}^* \in \mathcal{A}(\mathbf{z}_1, \mathbf{z}_2)} D(\mathbf{z}_1, \mathbf{z}_2, \mathbf{a}^*) \\ &= \min_{\mathbf{a}^* \in \mathcal{A}(\mathbf{z}_1, \mathbf{z}_2)} D(\mathbf{z}_2, \mathbf{z}_1, \mathbf{a}^*) = L(\mathbf{z}_2, \mathbf{z}_1) \end{aligned}$$

Finally let's prove that OTD satisfies triangle inequality, that is:

$$L(\mathbf{z}_1, \mathbf{z}_2) + L(\mathbf{z}_2, \mathbf{z}_3) \geq L(\mathbf{z}_1, \mathbf{z}_3), \quad (30)$$

where $\mathbf{z}_1$, $\mathbf{z}_2$ and $\mathbf{z}_3$ are three sequences. This property could be proved intuitively. Suppose that the operations on $\mathbf{z}_1$ with minimal costs to make $\mathbf{z}_1$ matched to $\mathbf{z}_2$ are denoted by $o_1, o_2, \ldots, o_{n_1}$, and those on $\mathbf{z}_2$ to make $\mathbf{z}_2$ matched to $\mathbf{z}_3$ are denoted by $o'_1, o'_2, \ldots, o'_{n_2}$. o_i could be a deletion, insertion or movement on a token. To make $\mathbf{z}_1$ matched to $\mathbf{z}_3$, one possible way, which is not necessarily the optimal, is to do $o_1, o_2, \ldots, o_{n_1}, o'_1, o'_2, \ldots, o'_{n_2}$ on $\mathbf{z}_1$. Since the total cost is the accumulation of the cost of each operation, and the operations on $\mathbf{z}_1$ above to make $\mathbf{z}_1$ matched to $\mathbf{z}_3$ might not be optimal, the triangle inequality equation (30) holds.

F. Approximate MBR Details

Our approximate consensus decoding algorithm is given as Algorithm 3. In the remainder of this section, we prove Theorem 1 from section 4.2, namely:

Theorem 1. *Given* $\{\mathbf{z}_m\}_{m=1}^M$*, if we define* $\mathbf{z}_\sqcup = \bigsqcup_{m=1}^M \mathbf{z}_m$*, then* $\exists \hat{\mathbf{z}} \subseteq \mathbf{z}_\sqcup$ *such that*

$$\textstyle\sum_{m=1}^M w_m L(\hat{\mathbf{z}}, \mathbf{z}_m) = \min_{\mathbf{z} \in \mathcal{Z}} \sum_{m=1}^M w_m L(\mathbf{z}, \mathbf{z}_m)$$

That is to say, there exists one subsequence of $\mathbf{z}_\sqcup$ *that achieves the minimum Bayes risk.*

Proof. Here we assume that there is only one type of event. Since the distances of different types of events are calculated separately, our conclusion is easy to be extended to the general case.

Suppose $\hat{\mathbf{z}}$ is an optimal decode, that is,

$$\sum_{m=1}^M w_m L(\mathbf{z}_m, \hat{\mathbf{z}}) = \min_{\mathbf{z} \in \mathcal{Z}} \sum_{m=1}^M w_m L(\mathbf{z}_m, \mathbf{z}).$$

[17] We abuse the notation $\mathbf{a}$, which we think could represent both the movement from $\mathbf{z}_1$ to $\mathbf{z}_2$ and from $\mathbf{z}_2$ to $\mathbf{z}_1$.

Algorithm 2 A Dynamic Programming Algorithm to Find Optimal Transport Distance

Input: proposal $\hat{\mathbf{z}}$; reference $\mathbf{z}^*$
Output: optimal transport distance d; alignment $\mathbf{a}$

procedure OTD($\hat{\mathbf{z}}, \mathbf{z}^*$)
  $d \leftarrow 0$; $\mathbf{a} \leftarrow$ empty collection $\{\}$
  for $k \leftarrow 1$ **to** K :
    $d^{(k)}, \mathbf{a}^{(k)} \leftarrow \text{ALIGN}(\hat{\mathbf{z}}^{(k)}, \mathbf{z}^{*(k)})$
    $d \leftarrow d + d^{(k)}$; $\mathbf{a} \leftarrow \mathbf{a} \cup \mathbf{a}^{(k)}$
  return $d, \mathbf{a}$
procedure ALIGN($\hat{\mathbf{z}}^{(k)}, \mathbf{z}^{*(k)}$)
  $\hat{I} \leftarrow |\hat{\mathbf{z}}^{(k)}|$; $I^* \leftarrow |\mathbf{z}^{*(k)}|$ ▷ $\hat{\mathbf{z}}^{(k)} = \hat{t}_1, \ldots, \hat{t}_{\hat{I}}$ *and* $\mathbf{z}^{*(k)} = t^*_1, \ldots, t^*_{I^*}$
  $\mathbf{D} \leftarrow$ zero matrix with $(\hat{I}+1)$ rows and (I^*+1) columns
  $\mathbf{P} \leftarrow$ empty matrix with $\hat{I}$ rows and I^* columns ▷ *back pointers*
  for $\hat{i} \leftarrow 1$ **to** $\hat{I}$: ▷ *transport reference of length* 0 *to proposal of length* $\hat{i}$
    $\mathbf{D}_{\hat{i},0} \leftarrow \mathbf{D}_{\hat{i}-1,0} + C_{\text{delete}}$ ▷ *delete* $\hat{t}_{\hat{i}}$ *(and prefixes are matched)*
  for $i^* \leftarrow 1$ **to** I^* : ▷ *transport preference of length* i^* *to proposal of length* 0
    $\mathbf{D}_{0,i^*} \leftarrow \mathbf{D}_{0,i^*} + C_{\text{insert}}$ ▷ *insert* $\hat{t}_{i^*} = t^*_{i^*}$ *to decode (and their prefixes are matched)*
  for $\hat{i} \leftarrow 1$ **to** $\hat{I}$: ▷ *proposal prefix of length* $\hat{i}$
    for $i^* \leftarrow 1$ **to** I^* : ▷ *to match reference of length* i^*
      $D_{\text{delete}} \leftarrow \mathbf{D}_{\hat{i}-1,i^*} + C_{\text{delete}}$ ▷ *if the event token at* $\hat{t}_{\hat{i}}$ *is deleted from* $\hat{\mathbf{z}}^{(k)}$
      $D_{\text{insert}} \leftarrow \mathbf{D}_{\hat{i},i^*-1} + C_{\text{insert}}$ ▷ *if an event token at* $t^*_{i^*}$ *is inserted to* $\hat{\mathbf{z}}^{(k)}$
      $D_{\text{move}} \leftarrow \mathbf{D}_{\hat{i}-1,i^*-1} + |\hat{t}_{\hat{i}} - t^*_{i^*}|$ ▷ *if the event at* $t^*_{i^*}$ *of* $\mathbf{z}^{*(k)}$ *is aligned to event at* $\hat{t}_{\hat{i}}$ *of* $\hat{\mathbf{z}}^{(k)}$
      $\mathbf{D}_{\hat{i},i^*} \leftarrow \min\{D_{\text{insert}}, D_{\text{delete}}, D_{\text{move}}\}$ ▷ *choose the edit that yields the shortest distance*
      $\mathbf{P}_{\hat{i},i^*} \leftarrow \text{argmin}_{e \in \{\text{insert,delete,move}\}} D_e$ ▷ e *represents the kind of edition*
  $\hat{i} \leftarrow \hat{I}$; $i^* \leftarrow I^*$; $\mathbf{a} \leftarrow$ empty collection$\{\}$
  while $\hat{i} > 0$ **and** $i^* > 0$: ▷ *back trace*
    if $\mathbf{P}_{\hat{i},i^*} =$ delete : ▷ *token* $\hat{t}_{\hat{i}}$ *is deleted.*
      $\hat{i} \leftarrow \hat{i} - 1$
    if $\mathbf{P}_{\hat{i},i^*} =$ insert : ▷ *a token at* $t^*_{i^*}$ *is inserted*
      $i^* \leftarrow i^* - 1$
    if $\mathbf{P}_{\hat{i},i^*} =$ move : ▷ *token* $t^*_{i^*}$ *is aligned to* $\hat{t}_{\hat{i}}$
      $\hat{i} \leftarrow \hat{i} - 1; i^* \leftarrow i^* - 1$
      $\mathbf{a} \leftarrow \mathbf{a} \cup \{(\hat{t}_{\hat{i}}, t^*_{i^*})\}$
  return $\mathbf{D}_{\hat{I},I^*}, \mathbf{a}$

If $\hat{\mathbf{z}} \subseteq \mathbf{z}_\sqcup$, the proof is done. If not, we can choose some $t_i \notin \mathbf{z}_\sqcup$. Let $t_l = \max\{t \in \mathbf{z}_\sqcup : t < t_i\}$ and $t_r = \min\{t \in \mathbf{z}_\sqcup : t > t_i\}$. (These sets are nonempty because $\mathbf{z}_\sqcup$ always contains the endpoints 0 and T.) We will show that if we move t_i around, as long as $t_i \in [t_l, t_r]$, the weighted optimal transport distance, i.e. $\sum_{m=1}^M w_m L(\mathbf{z}_m, \hat{\mathbf{z}})$, will neither increase nor decrease.

Suppose $\hat{\mathbf{a}} = \text{argmin}_{\mathbf{a}_m \in \mathcal{A}(\mathbf{z}_m, \hat{\mathbf{z}})} \sum_{m=1}^M w_m D(\mathbf{z}_m, \hat{\mathbf{z}}, \mathbf{a}_m)$. Let's use $r(t)$ to indicate the weighted transport distance of $\hat{\mathbf{z}}$ with fixed alignment if we move t_i to t, that is,

$$r(t) \stackrel{\text{def}}{=} \sum_{m=1}^M w_m D(\mathbf{z}_m, \hat{\mathbf{z}}(t), \hat{\mathbf{a}}),$$

where $\hat{\mathbf{z}}(t)$ is the sequence $\hat{\mathbf{z}}$ with t_i moved to t. Because $\hat{\mathbf{z}}(t_i)$ is an optimal decode, and $\hat{\mathbf{a}}$ is the optimal alignment for $\hat{\mathbf{z}}(t_i)$, we should have

$$r(t_i) = \min_t r(t).$$

Note that the transport distance is comprised of three parts: deletion, insertion and alignment costs. Since every $\hat{\mathbf{a}}$ is fixed, if we change t, only the alignment cost that related to token t will affect $r(t)$. This part of $r(t)$ is linear to t, since we have a constraint $t \in [t_l, t_r]$, which guarantees that it will not cross any other tokens in $\mathbf{z}_\sqcup$.

Since $r(t)$ is linear to $t \in [r_l, t_r]$ and $r(t)$ gets minimized at $t_i \in (t_l, t_r)$, we conclude that

$$r(t) = r(t_i) = \text{Const}, \forall t \in [t_l, t_r].$$

Since $r(t)$ is the upper bound of the weighted optimal transport distance $\sum_{m=1}^M w_m L(\mathbf{z}_m, \hat{\mathbf{z}}(t))$, which also gets the

Algorithm 3 Approximate Consensus Decoding

Input: collection of weighted particles $\mathcal{Z}_M = \{(\mathbf{z}_m, w_m)\}_{m=1}^M$
Output: consensus sequence $\hat{\mathbf{z}}$ with low $\sum_{m=1}^M w_m L(\hat{\mathbf{z}}, \mathbf{z}_m)$

procedure APPROXMBR($\mathcal{Z}_M$)
  $\hat{\mathbf{z}} \leftarrow$ empty sequence
  for $k = 1$ **to** K :
    $\hat{\mathbf{z}}^{(k)} \leftarrow$ DECODEK($\{(\mathbf{z}_m^{(k)}, w_m)\}_{m=1}^M$); $\hat{\mathbf{z}} \leftarrow \hat{\mathbf{z}} \sqcup \hat{\mathbf{z}}^{(k)}$ ▷ *decode for type-k by calling* DECODEK
  return $\hat{\mathbf{z}}$
procedure DECODEK($\mathcal{Z}_M$)
  ▷ $\mathcal{Z}_M$ *actually means* $\mathcal{Z}_M^{(k)} = \{(\mathbf{z}_m^{(k)}, w_m)\}_{m=1}^M$ *throughout the procedure;* $\mathbf{z}_m$ *is constant*
  $\mathbf{z} \leftarrow \text{argmax}_{\mathbf{z} \in \{\mathbf{z}_m\}_{m=1}^M} w_m$ ▷ *init decode as highest weighted particle and it is global*
  repeat
    for $m = 1$ **to** M : ▷ **Align Phase**
      $d_m, \mathbf{a}_m \leftarrow$ ALIGN($\mathbf{z}, \mathbf{z}_m$) ▷ *call method in Algorithm 2;* $d_m, \mathbf{a}_m$ *are global*
    $r_{\text{min}} \leftarrow \sum_m w_m d_m$ ▷ *track the risk of current* $\mathbf{z}$
    $\mathbf{z}, \{d_m, \mathbf{a}_m\}_{m=1}^M \leftarrow$ MOVE($\mathbf{z}, \{\mathbf{z}_m, d_m, \mathbf{a}_m\}_{m=1}^M$) ▷ *see Algorithm 4*
    $\mathbf{z}, \{d_m, \mathbf{a}_m\}_{m=1}^M \leftarrow$ DELETE($\mathbf{z}, \{\mathbf{z}_m, d_m, \mathbf{a}_m\}_{m=1}^M$) ▷ *see Algorithm 4*
    $\mathbf{z}, \{d_m, \mathbf{a}_m\}_{m=1}^M \leftarrow$ INSERT($\mathbf{z}, \{\mathbf{z}_m, d_m, \mathbf{a}_m\}_{m=1}^M$) ▷ *see Algorithm 4*
  until $\sum_{m=1}^M w_m d_m = r_{\text{min}}$ ▷ *risk stops decreasing*
  return $\mathbf{z}$

same minimal value at $t_i \in (t_l, t_r)$ as $r(t)$, we could conclude that $\forall t \in [t_l, t_r]$:

$$\sum_{m=1}^M w_m L(\mathbf{z}_m, \hat{\mathbf{z}}(t)) = \sum_{m=1}^M w_m L(\mathbf{z}_m, \hat{\mathbf{z}}(t_i)) = \text{Const}$$

Therefore we could move token t_i to either t_l or t_r without increasing the Bayes risk. We could do this movement for each $t_i \notin \mathbf{z}_\sqcup$ to get a new decode $\hat{\mathbf{z}} \subseteq \mathbf{z}_\sqcup$, which is also an optimal decode.

□

G. Experimental Details

In this section, we elaborate on the details of data generation, processing, and experimental results.

In all of our experiments, the distribution p_{model} is trained on the complete (uncensored) version of the training data. The system is then asked to complete the incomplete (censored) version of the test (or dev) data. For particle smoothing, the proposal distribution is trained using both the complete and incomplete versions of the training data, as explained at the end of section 3.2.1. We used the Adam algorithm with its default settings (Kingma & Ba, 2015). Adam is a stochastic gradient optimization algorithm that continually adjusts the learning rate in each dimension based on adaptive estimates of low-order moments. Each training example for Adam is a complete event stream $\mathbf{x} \sqcup \mathbf{z}$ over some time interval $[0, T)$. We stop training early when we detect that log-likelihood has stopped increasing on the held-out development dataset. We do no other regularization.

G.1. Dataset Statistics

Table 1 shows statistics about each dataset that we use in this paper.

G.2. Training Details

We used single-layer LSTMs (Hochreiter & Schmidhuber, 1997), selected the number D of hidden nodes of the left-to-right LSTM, and then D' of the right-to-left one from a small set $\{16, 32, 64, 128, 256, 512, 1024\}$ based on the performance on the dev set of each dataset. The best-performing (D, D') pairs are $(256, 128)$ on Synthetic, $(256, 256)$ on Elevator $(256, 256)$ on NYC Taxi, but we empirically found that the model performance is robust to these hyperparameters. For the chosen (D, D') pair on each dataset, we selected β based on the performance on the dev set, and $\beta = 1.0$ yields the best performance across all the datasets we use. For learning, we used Adam with its default settings (Kingma & Ba, 2015).

Our Monte Carlo integral estimates are in fact unbiased (Appendix C.3). As a result, our stochastic gradient estimate is also unbiased, as required (assuming that the complete data is distributed according to p_{model}). Why? Since $\beta = 1$, our stochastic gradient is simply equation (6). No particle filtering or smoothing is used to estimate equation (6), because we train it using complete data, as explained in the last long paragraph of section 3.2.1. The

Algorithm 4 Subroutines for Approximate Consensus Decoding

procedure MOVE($\mathbf{z}, \{\mathbf{z}_m, d_m, \mathbf{a}_m\}_{m=1}^M$) ▷ **Move Phase**
  for t **in** $\mathbf{z}$:
    for $t' \in \{t' : (t', t) \in \bigcup_{m=1}^M \mathbf{a}_m\}$: ▷ *may replace t with t' which is aligned to t*
      $(\forall m) d'_m \leftarrow d_m$
      for $(t'', m) \in \{(t'', m) : (t'', t) \in \mathbf{a}_m, m \in \{1, \dots, M\}\}$:
        $d'_m \leftarrow d'_m - |t'' - t| + |t'' - t'|$
      if $\sum_m w_m d'_m < \sum_m w_m d_m$:
        $(\forall m) d_m \leftarrow d'_m$; $t \leftarrow t'$ ▷ *t move to t' for lower risk*
  return $\mathbf{z}, \{d_m, \mathbf{a}_m\}_{m=1}^M$
procedure DELETE($\mathbf{z}, \{\mathbf{z}_m, d_m, \mathbf{a}_m\}_{m=1}^M$) ▷ **Delete Phase**
  for t **in** $\mathbf{z}$: ▷ *may delete this event*
    for $m = 1$ **to** M : ▷ *update each d_m*
      if $\exists t' \in \mathbf{z}_m$ **and** $(t', t) \in \mathbf{a}_m$: ▷ *find the only, if any, $t' \in \mathbf{z}_m$ that is aligned to t*
        ▷ *if we delete t and its alignment (t', t), d_m decreases by the alignment cost (because we do not need to align it)*
        ▷ *but increases by an insertion cost (because we need to insert an event at t to match $\mathbf{z}_m$)*
        $d'_m \leftarrow d_m + C_{\text{insert}} - |t' - t|$
      else ▷ *otherwise, this event has been deleted when matching with $\mathbf{z}_m$*
        $d'_m \leftarrow d_m - C_{\text{delete}}$ ▷ *we do not need to pay deletion cost when matching with $\mathbf{z}_m$ if we do not have this event at t in $\mathbf{z}$*
      if $\sum_m w_m d'_m < \sum_m w_m d_m$:
        delete t from $\mathbf{z}$; $(\forall m)$ delete (t', t) from $\mathbf{a}_m$; $d_m \leftarrow d'_m$
  return $\mathbf{z}, \{d_m, \mathbf{a}_m\}_{m=1}^M$
procedure INSERT($\mathbf{z}, \{\mathbf{z}_m, d_m, \mathbf{a}_m\}_{m=1}^M$) ▷ **Insert Phase**
  repeat
    $t \leftarrow$ None, $\Delta \leftarrow -\infty$
    for $t_c \in \bigsqcup_m \mathbf{z}_m$ such that $t_c \notin \mathbf{z}$: ▷ *may insert t_c if it is not in $\mathbf{z}$ yet*
      for $m = 1$ **to** M :
        $\mathbf{z}'_m \leftarrow \{t' : \forall t'', (t'', t') \notin \mathbf{a}_m \text{ and } t' \in \mathbf{z}_m\}$ ▷ *find t' in $\mathbf{z}_m$ that is not aligned yet*
        if $\mathbf{z}'$ is not empty **and** $\min_{t' \in \mathbf{z}'_m} |t' - t_c| < C_{\text{insert}} + C_{\text{delete}}$: ▷ *if there is any that is close enough to t_c*
          $d'_m \leftarrow d_m - C_{\text{insert}} + \min_{t' \in \mathbf{z}'_m} |t' - t_c|$; $\mathbf{a}'_m \leftarrow \mathbf{a}_m \cup \{(t_c, t')\}$ ▷ *align the closest one to t_c*
        else
          $d'_m \leftarrow d_m + C_{\text{delete}}$; $\mathbf{a}'_m \leftarrow \mathbf{a}_m$
      if $\sum_m w_m d_m - \sum_m w_m d'_m > \Delta$:
        $t \leftarrow t_c$; $\Delta \leftarrow \sum_m w_m d_m - \sum_m w_m d'_m$
    if $\Delta > 0$:
      $\mathbf{z} \leftarrow \mathbf{z} \sqcup \{t\}$; $(\forall m) \mathbf{a}_m \leftarrow \mathbf{a}'_m$; $d_m \leftarrow d'_m$
  until $\Delta \leq 0$
  return $\mathbf{z}, \{d_m, \mathbf{a}_m\}_{m=1}^M$

DATASET	K	# OF EVENT TOKENS			SEQUENCE LENGTH		
		TRAIN	DEV	TEST	MIN	MEAN	MAX
SYNTHETIC	4	≈ 74967	≈ 7513	≈ 7507	10	≈ 15	20
NYCTAXI	10	157916	15826	15808	22	32	38
ELEVATOR	10	313043	31304	31206	235	313	370

Table 1. Statistics of each dataset. We write "$\approx N$" to indicate that N is the average value over multiple datasets of one kind (synthetic); the variance is small in each such case.

only randomness is the integral over $[0, T)$ (similar to the one in equation (19)) that is required to estimate the term $\log q(\mathbf{z} \mid \mathbf{x})$ in equation (6): as just noted, this integral estimate is unbiased.

It is true that if $\beta < 1$, we would compute the exclusive KL gradient using particle filtering or smoothing with M particles, and this would introduce bias in the gradient. Nonetheless, since the bias vanishes as $M \to \infty$, it would be possible to restore a theoretical convergence guarantee by increasing M at an appropriate rate as SGD proceeds (Spall, 2005, page 107).[18]

G.3. Details of the Synthetic Datasets

Each of the ten neural Hawkes processes has its parameters sampled from $\mathrm{Unif}[-1.0, 1.0]$. Then a set of event sequences is drawn from each of them via the plain vanilla thinning algorithm (Mei & Eisner, 2017). For each of the ten synthetic datasets, we took $K = 4$ as the number of event types. To draw each event sequence, we first chose the sequence length I (number of event tokens) uniformly from $\{11, 12, \ldots, 20\}$ and then used the thinning algorithm to sample the first I events over the interval $[0, \infty)$. For subsequent training or testing, we treated this sequence (appropriately) as the complete set of events observed on the interval $[0, T)$ where $T = t_I$, the time of the last generated event.

We generate 5000, 500 and 500 sequences for each training, dev, and test set respectively. For the missingness mechanism: in the deterministic settings, we censor all events of type 3 and 4—in other words, we set $\rho_1 = \rho_2 = 0$ and $\rho_3 = \rho_4 = 1$; in the stochastic settings, we set $\rho_k = 0.5$ for all k.

G.4. Elevator System Dataset Details

We examined our method in a simulated 5-floor building with 2 elevator cars. During a typical afternoon down-peak rush hour (when passengers go from floor-2,3,4,5 down to the lobby), elevator cars travel to each floor and pick up passengers that have (stochastically) arrived there according to a traffic profile (Bao et al., 1994). Each car will also avoid floors that already are or will soon be taken care of by the other. Having observed when and where car-1 has stopped (to pick up or drop off passengers) over this hour, we are interested in when and where car-2 has stopped during the same time period. In this dataset, each event type is a tuple of (car number, floor number) so there are $K = 10$ in total in this simulated 5-floor building with 2 elevator cars.

Passenger arrivals at each floor are assumed to follow a inhomogeneous Poisson process, with arrival rates that vary during the course of the day. The simulations we use follows a human-recorded traffic profile (Bao et al., 1994) which dictates arrival rates for every 5-minute interval during a typical afternoon down-peak rush hour. Table 2 shows the mean number of passengers (who are going to the lobby) arriving at floor-2,3,4,5 during each 5-minute interval.

We simulated the elevator behavior following a naive baseline strategy documented in Crites & Barto (1996).[19] In details, each car has a small set of primitive actions. If it is stopped at a floor, it must either "move up" or "move down". If it is in motion between floors, it must either "stop at the next floor" or "continue past the next floor". Due to passenger expectations, there are two constraints on these actions: a car cannot pass a floor if a passenger wants to get off there and cannot turn until it has serviced all the car buttons in its current direction. Three additional action constraints were made in an attempt to build in some primitive prior knowledge: 1) a car cannot stop at a floor unless someone wants to get on or off there; 2) it cannot stop to pick up passengers at a floor if another car is already stopped there; 3) given a choice between moving up and down, it should prefer moving up (since the down-peak traffic tends to push the cars toward the bottom of the building). Because of this last constraint, the only real choices left to each car are the stop and continue actions, and the

[18] SGD methods succeed, both theoretically and practically, with even high-variance estimates of the batch gradient (e.g., where each stochastic estimate is derived from a single randomly chosen training example). Thus, one should be fine with a noisy sampling-based gradient as long as it is unbiased.

[19] We rebuilt the system in Python following the original Fortran code of Crites & Barto (1996).

baseline strategy always chooses to continue. The actions of the elevator cars are executed asynchronously since they may take different amounts of time to complete.

We repeated the (one-hour) simulation 700 times to collect the event sequences, each of which has around 300 time-stamped records of which car stops at which floor. We randomly sampled disjoint train, dev and test sets with 500, 100 and 100 sequences respectively.

For the missingness mechanism: in the deterministic settings, we set $\rho_k = 0$ for $k = 1, \ldots, 5$ and $\rho_k = 1$ for $k = 6, \ldots, 10$ (meaning that the events (of arriving at floor 1, 2, ..., 5) of car 1 are all observed, but those of car 2 are not); in the stochastic settings, we set $\rho_k = 0.5$ for all k.

G.5. New York City Taxi Dataset Details

The New York City Taxi dataset (section 5.2) includes 189,550 taxi pick-up and drop-off records in the city of New York in 2013. Each record has its medallion ID, driver license and time stamp. Each combination of medallion ID and driver license naturally forms a sequence of time-stamped pick-up and drop-off events. Following the processing recipe of previous work (Du et al., 2016), we construct shorter sequences by breaking each long sequence wherever the temporal gap between a drop-off event and its following pick-up event is larger than six hours. Then the left boundary of this gap is treated as the EOS of the sequence before it, while the right boundary is set as the BOS of the following sequence.

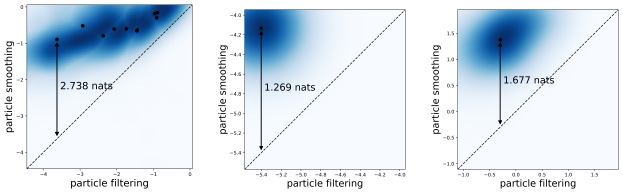
We randomly sampled a month from 2013 and then randomly sampled disjoint train, dev and test sets with 5000, 500 and 500 sequences respectively from that month.

In this dataset, each event type is a tuple of (location, action). The location is one of the 5 boroughs {Manhattan, Brooklyn, Queens, The Bronx, Staten Island}. The action can be either pick-up or drop-off. Thus, there are $K = 5 \times 2 = 10$ event types in total.

For the missingness mechanism: in the deterministic settings, we set $\rho_k = 0$ for $k = 1, \ldots, 5$ and $\rho_k = 1$ for $k = 6, \ldots, 10$ (which means that all drop-off events but no pick-up events are observed); in the stochastic settings, we set $\rho_k = 0.5$ for all k.

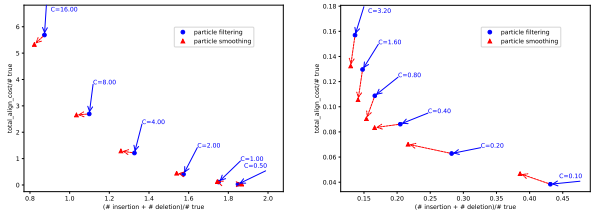
G.6. Experiments with Deterministic Missingness Mechanisms

We show our experimental results for the deterministic missingness mechanisms in Figures 4 and 5.



(a) Synthetic (b) Elevator System (c) NYC Taxi

Figure 4. Scatterplots with a deterministic missingness mechanism. Again, the method works, with very similar qualitative behavior to Figure 2.



(a) Elevator System (b) NYC Taxi

Figure 5. Optimal transport distance results with a deterministic missingness mechanism. Again, the method works, with very similar qualitative behavior to Figure 3.

G.7. Sensitivity Experiment Details

Figure 6 displays the optimal transport distance with various values of ρ: our particle smoothing method consistently outperforms the filtering baseline.

G.8. Wall-Clock Runtime Details

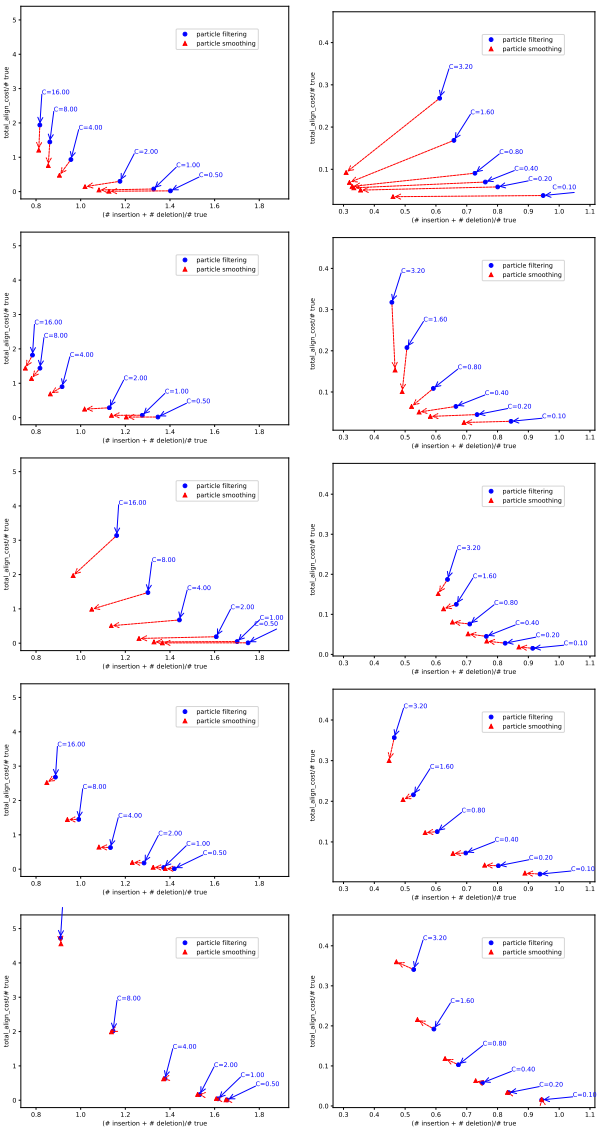
A given run of particle smoothing begins by drawing $O(I)$ time points from $\mathrm{Unif}([0, T))$, where I is the number of observed events. All particles are evaluated using integrals that are estimated by evaluating the function at these time points (Appendix C.3).

The theoretical runtime complexity is $O(MI)$ because drawing a particle requires $O(I)$ time—the outer loop over time steps (line 7 of Algorithm 1)—and we draw M particles in total—the inner loop over particles (line 8 in Algorithm 1). Our GPU implementation (which we will release) parallelizes the inner loop over particles. We sample 50 particles in parallel in these experiments, but we have tested with 1000 particles in parallel as well. So this is not a real problem with a GPU.

We reported experiments that we performed to demonstrate the practicality. On average, drawing an ensemble of 50 particles takes 5 seconds per example on the synthetic datasets (average length 15 events), 12 seconds per example on the NYC Taxi dataset (average length 32 events) and 100 seconds per example on the Elevator System dataset (average length 313 events)—that is, 300-400 milliseconds

START TIME (MIN)	00	05	10	15	20	25	30	35	40	45	50	55
MEAN # PASSENGER	1	2	4	4	18	12	8	7	18	5	3	2

Table 2. The Down-Peak Traffic Profile



(a) Elevator System (b) NYC Taxi

Figure 6. Optimal transport distance results with varying missingness rate ρ. Rows (top-down) are results with $\rho = 0.1, 0.3, 0.5, 0.7, 0.9$. As we can see, our particle smoothing consistently outperforms the filtering baseline with different ρ, although no clear trend with increasing ρ is found on either dataset.

per event. Such speeds are acceptable in many incomplete data applications, compared to the cost of collecting complete data—all the applications in section 1 involve real-time decision making at a human timescale.

H. Monte Carlo EM

We normally assume (section 3.2.1) that some complete sequences are available for training the neural Hawkes process models. If incomplete sequences are also available, our particle smoothing method can be used to (approximately) impute the missing events, which yields additional complete sequences for training. Indeed, if we are willing to make a MAR assumption (Little & Rubin, 1987), then we can do imputation without modeling the missingness mechanism. Training on such imputed sequences is an instance of **Monte Carlo expectation-maximization (MCEM)** (Dempster et al., 1977; Wei & Tanner, 1990; McLachlan & Krishnan, 2007), with particle smoothing as the Monte Carlo E-step, and makes it possible to train with incomplete data only.

In the more general MNAR scenario, we can extend the E-step to consider the not-at-random missingness mechanism (see equation (3) below), but then we need both complete and incomplete sequences at training time in order to fit the parameters of the missingness mechanism (unless these parameters are already known) jointly with those of the neural Hawkes process. Although training with incomplete data is out of the scope of our experiments, we describe the methods here and provide MCEM pseudocode.

In this case, we would like to know the (marginal) probability of the observed data $\mathbf{x}$ under the target distribution p:

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z}) p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z}) \tag{31}$$

If we propose $\mathbf{z}$ from $q(\mathbf{z} \mid \mathbf{x})$, then it can be rewritten as:

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z}) p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z}) \frac{q(\mathbf{z} \mid \mathbf{x})}{q(\mathbf{z} \mid \mathbf{x})} \tag{32a}$$

$$= \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})} \left[\frac{p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z}) p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z})}{q(\mathbf{z} \mid \mathbf{x})} \right] \tag{32b}$$

Given a finite number M of proposed particles $\{\mathbf{z}_m\}_{m=1}^M$, this expectation can be estimated with empirical average:

$$p(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} \frac{p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z}_m) p_{\text{miss}}(\mathbf{z}_m \mid \mathbf{x} \sqcup \mathbf{z}_m)}{q(\mathbf{z}_m \mid \mathbf{x})} \tag{33}$$

and it is obvious that

$$\log p(\mathbf{x}) \geq \frac{1}{M} \sum_{m=1}^{M} (b_m - \log q(\mathbf{z}_m \mid \mathbf{x})) \tag{34a}$$

$$b_m = \log p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z}_m) + \log p_{\text{miss}}(\mathbf{z}_m \mid \mathbf{x} \sqcup \mathbf{z}_m) \tag{34b}$$

where the right-hand-side (RHS) term of equation (34a) is the **Evidence Lower Bound (ELBO)** that we would maximize in order to maximize the log-likelihood.

The MCEM algorithm is composed of two steps:

E(xpectation)-step We train the proposal distribution $q(\mathbf{z} \mid \mathbf{x})$ using the method in section 3.2.1 and then sample M weighted particles from $q(\mathbf{z} \mid \mathbf{x})$ by calling Algorithm 1.

M(aximization)-step We train the neural Hawkes process $p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z})$ by maximizing the ELBO (equation (34a)).

Note that in the MAR case, $p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z})$ is constant of $\mathbf{z}$ so the it can be omitted from the formulation (and thus the algorithms). Also note that, for particle filtering, the proposal distribution $q(\mathbf{z} \mid \mathbf{x})$ is only part of $p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z})$ so we do not need to train $q(\mathbf{z} \mid \mathbf{x})$ at the E-step.

Maximum-likelihood estimation remains sensible in the MNAR case provided that we know one of the distributions p_{model} or p_{miss}, in which case we can use EM to estimate the other distribution.

(1) If p_{miss} is known and fixed, as in our experiments, this gives a minor variant of ordinary EM. Ordinary EM makes the MAR assumption that the p_{miss} factor of equation (1) can be ignored. However, if we know p_{miss}, we can incorporate it rather than ignoring it; then it need not satisfy the MAR assumption.

(2) Conversely, if p_{model} is known and fixed because we estimated it from a sufficient quantity of *complete* data, then we can use incomplete data to learn the MNAR missingness distribution p_{miss}. This setting would even lets us learn contextual missingness mechanisms in which the probability that an event is censored depends not only on the event itself, but also on the surrounding events and whether they are censored. For example, one could try to fit p_{miss} with an LSTM model or a BiLSTM-CRF model (Huang et al., 2015) that performs structured joint prediction of the missingness of all events in the sequence. Extending that method to use continuous-time LSTMs would allow it to take timing into account.

The E step of Monte Carlo EM uses the current guesses of p_{model} and/or p_{miss} to sample from the posterior distribution $p(\mathbf{z} \mid \mathbf{x})$ of the missing values. That posterior is uncontroversially defined by the simple Bayesian formula (1). Notice that even if p_{model} and p_{miss} were *both* unknown, we could still run MCEM to locally maximize the likelihood $p(\mathbf{x})$, but unfortunately the parameters would be unidentifiable in this case. Thus, there would be many missing-data models with the same likelihood, as explained in Appendix A, and they would make different predictions of $\mathbf{z}$.