SMCP^3: SMC with Probabilistic Program Proposals

Alexander K. Lew[1,*] **George Matheos**[2,*] **Tan Zhi-Xuan**[1] **Matin Ghavamizadeh**[1]
Nishad Gothoskar[1] **Stuart Russell**[2] **Vikash K. Mansinghka**[1]
[1]MIT [2]UC Berkeley [*]Equal contribution

Abstract

There is a widespread need for sound, flexible frameworks for Monte Carlo inference. This paper introduces SMCP^3, a sequential Monte Carlo framework that broadens the class of strategies practitioners can employ to update particles from iteration to iteration, relative to existing frameworks like resample-move SMC (Gilks & Berzuini, 2001) and SMC samplers (Del Moral et al., 2006). In SMCP^3, proposal kernels can be general probabilistic programs, which differ from traditional proposal densities in that they may sample many auxiliary variables, and may apply deterministic post-processing to calculate a proposed update. We have implemented our framework in the Gen probabilistic programming platform: given probabilistic programs that specify target distributions, forward kernels, and reverse kernels, our implementation fully automates the sound computation of incremental importance weights. To illustrate the effectiveness of SMCP^3 algorithms, we apply our framework in two domains. First, we use it for online state-estimation, using proposal programs based on Langevin ascent to reduce the bias in log marginal likelihood estimates relative to resample-move SMC with Langevin rejuvenation. Second, we demonstrate an SMCP^3 algorithm that yields more robust online clustering in Dirichlet process mixture models than strong SMC baselines.

1 Introduction

This paper aims to enable probabilistic programmers to easily implement custom sequential Monte Carlo (SMC) algorithms that draw on and extend sophisticated techniques such as MCMC rejuvenation (24) and paired forward and reverse Markov kernels (11, 14). The framework we introduce, called SMCP^3, automates the implementation of particle updates for proposals specified by general probabilistic programs—including proposal kernels that sample auxiliary variables, lack tractable marginal probability densities, and incorporate deterministic transformations—by applying techniques from probabilistic programming. SMCP^3 simultaneously provides automation for time-consuming, error-prone aspects of algorithm implementation and expands the range of proposals that algorithm designers can easily experiment with. It thus has the potential to bring modern SMC methods to the rapidly growing audience of probabilistic programmers, and to enable SMC experts to explore more complex techniques.

We illustrate the value of custom SMC with probabilistic program proposals with examples from state-space modeling and mixture modeling (30, 53). We show that custom SMCP^3 samplers can achieve better marginal likelihood estimates compared to strong SMC baselines, on both synthetic and real-world datasets. For state-space models, we exhibit an SMCP^3 algorithm that combines gradient-based proposals with a simple reverse kernel to yield improved importance weights relative to resample-move approaches (24). For mixture models, we give an SMCP^3 algorithm that uses a split-merge proposal with a custom reverse kernel to improve over locally optimal "Gibbs-style" SMC updates for each new datapoint.

Contributions. This paper contributes:

1. The SMCP^3 mathematical framework (Section 2), which shows how to compute valid weights when proposals are general probabilistic computations that may not admit tractable marginal densities.
2. An algorithm for automating SMCP^3 in probabilistic programming systems, and an implementation in Gen (Section 3).
3. Example SMCP^3 algorithms for state-space and

Preprint. To appear at AISTATS 2023.

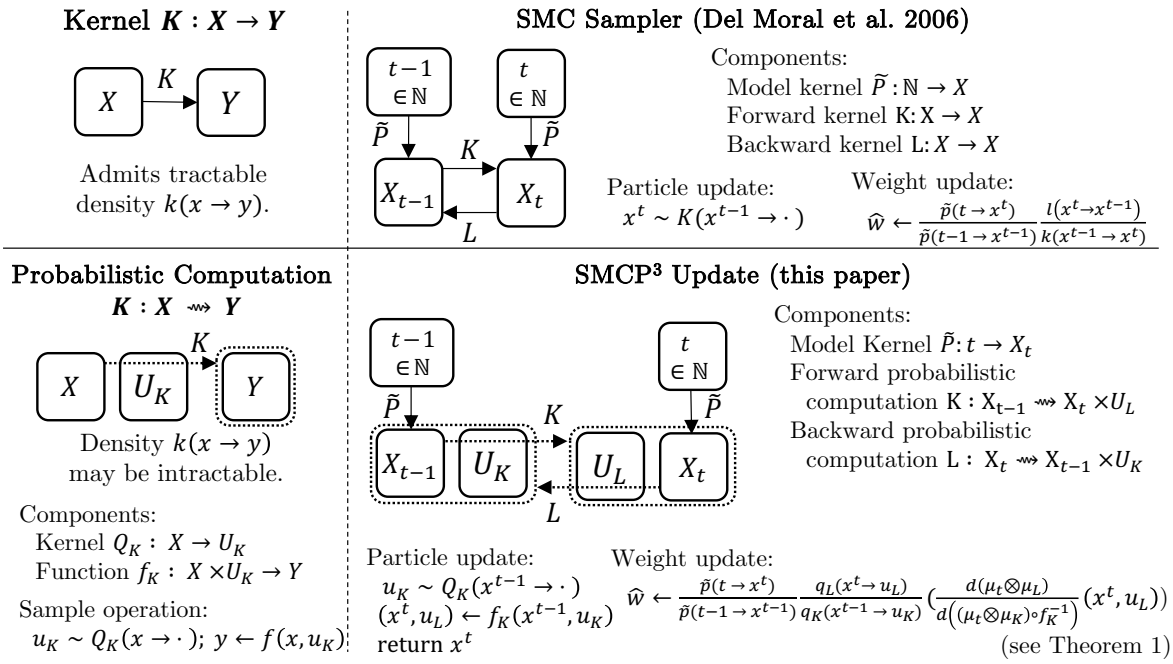


Figure 1: **Left:** In most treatments of SMC, particle update proposals are specified as *kernels* $K : X \to Y$ with analytically tractable densities $k(x \to y)$. To support probabilistic program proposals, we formalize SMCP³ in terms of more general *probabilistic computations* $K : X \rightsquigarrow Y$, which sample auxiliary variables, then transform them deterministically. **Right:** SMCP³ generalizes Del Moral et al. (12, 14), by supporting K and L kernels specified as probabilistic computations, rather than kernels with densities. In Section 3, we show how to automate the computation of the incremental weight $\hat{w}$ when $\tilde{P}$, K and L are given as probabilistic programs.

mixture models, which exploit SMCP³'s new degrees of freedom and outperform strong resample-move and particle filter baselines (Section 4).

2 The SMCP³ Algorithm

Consider a sequence $\tilde{P}_1, \ldots, \tilde{P}_T$ of unnormalized measures, defined over corresponding measurable spaces $X_1, \ldots, X_T$.[1] We assume reference measures μ_t on X_t (e.g., the Lebesgue measure, if X_t is $\mathbb{R}^n$) with respect to which each $\tilde{P}_t$ has a density, $\tilde{p}_t$. The goal of SMC is to develop weighted particle approximations to each $\tilde{P}_t$, that is, collections $\{(x_i^t; w_i^t)\}_{i=1}^N$ such that integrals $\int_{X_t} f(x)\tilde{P}_t(dx)$ can be approximated unbiasedly, and ideally with low variance, by $\hat{I} = \sum_{i=1}^N w_i^t f(x_i^t)$.[2]

Sequential Monte Carlo algorithms (1, 6, 11, 13, 17, 49) successively approximate each measure in this sequence, using the particles for the previous measure as starting points to form the particles for the next measure. After *initializing* a particle collection $\{(x_i^1; w_i^1)\}_{i=1}^N$ via importance sampling targeting $\tilde{P}_1$, SMC alternates between *resampling* steps (16), in which promising particles are selected to serve as the basis for future inferences, and *updating* steps, in which resampled particles are independently updated, to form an approximation to the next target.

Algorithm 1 presents our variant of SMC, called SMCP³. The key mathematical novelty in SMCP³ is in how particle updates are implemented, and how incremental importance weights are computed. The aim of this new development is to enable us, in Section 3, to interpret user-specified probabilistic programs as particle update strategies, and automate the necessary sampling and weight calculations. To that end, we formulate updates not in terms of proposal densities but rather *probabilistic computations* (Fig. 1, left), which may sample many auxiliary variables, then apply deterministic post-processing to generate an update.

[1] Readers may be familiar with particle filters (in which $X_t = S^t$ for some state space S), or Del Moral et al. (14)'s SMC samplers (in which the state space X_t does not vary in time). In our setting, the state spaces may vary arbitrarily.

[2] In Bayesian applications, X_t is the latent space of a generative model, and $\tilde{P}_t$ is the unnormalized posterior obtained by conditioning the model on observed data. To estimate a normalized posterior expectation, *self-normalization* is required, yielding a biased but consistent estimator, $\frac{\sum w_i^t f(x_i^t)}{\sum w_i^t}$. The denominator is an unbiased estimate of the normalization constant $\int \tilde{P}_t(dx)$, also known as the *evidence* or *marginal likelihood* of the observed data.

Algorithm 1 SMCP^3

Require: Sequence of target densities $\tilde{p}_t$ w.r.t. $\quad_t$
Require: Initial proposal Q_1 w/ density q_1 w.r.t. $\quad_1$
Require: Forward proposals $K_t : X_{t-1} \quad X_t \quad U_{L_t}$
Require: Reverse proposals $L_t : X_t \quad X_{t-1} \quad U_{K_t}$
Require: Number of particles N
Ensure: $(x_i^t; w_i^t)_{i=1}^N$ properly weighted for $\tilde{P}_t$

1: . Initialize particles with importance sampling
2: **for** $i = 1;:::;N$ **do**
3: $\quad x_i^1 \quad Q_1$
4: $\quad w_i^1 \quad \frac{\tilde{p}_1(x_i^1)}{q_1(x_i^1)}$
5: **end for**
6: . Main SMC loop
7: **for** $t = 2;:::;T$ **do**
8: $\quad$. Draw ancestor indices
9: $\quad (a_i^t)_{i=1}^N \quad$ Categorical$([w_1^{t-1};:::;w_N^{t-1}])$
10: $\quad$. Compute average weight
11: $\quad W^{t-1} \quad \frac{\sum_{i=1}^N w_i^{t\ 1}}{N}$
12: $\quad$ **for** $i = 1;:::;N$ **do**
13: $\quad\quad \tilde{x} \quad x_{a_i^t}^{t-1}$
14: $\quad\quad$. Draw auxiliary randomness u_{K_t}
15: $\quad\quad u_{K_t} \quad Q_{K_t}(\tilde{x} \ ! \ )$
16: $\quad\quad$. Apply deterministic transformation
17: $\quad\quad (x_i^t; u_{L_t}) \quad f_{K_t}(\tilde{x}; u_{K_t})$
18: $\quad\quad$. Compute density ratio
19: $\quad\quad \hat{w}_i^t \quad \frac{\tilde{p}_t(x_i^t) q_{L_t}(x_i^t \to u_{L_t})}{\tilde{p}_{t\ 1}(\tilde{x}) q_{K_t}(\tilde{x} \to u_{K_t})}$
20: $\quad\quad$. Incorporate change-of-variables factor
21: $\quad\quad \hat{w}_i^t \quad \hat{w}_i^t \ \frac{d(\mu_t \otimes \mu_{L_t})}{d((\mu_{t\ 1} \otimes \mu_{K_t}) \circ f_{K_t}^{\ 1})}(x_i^t; u_{L_t})$
22: $\quad\quad$. Weight update
23: $\quad\quad w_i^t \quad W^{t-1} \hat{w}_i^t$
24: $\quad$ **end for**
25: **end for**

Definition 1 (Probabilistic computation)**.** A *probabilistic computation* $K : X \quad Y$ between measurable spaces X and Y is a tuple $(U_K; Q_K; f_K)$, where:

- $U_K = (jU_K j; U_K; \quad_K)$ is a measure space of auxiliary randomness,
- $Q_K : X \ ! \ U_K$ is a probability kernel with density q_K with respect to $\quad_K$, and
- $f_K : X \quad U_K \ ! \ Y$ is a measurable map.

To run a probabilistic computation, we sample $u \quad Q_K(x \ ! \ )$, then compute $y = f_K(x; u)$. Importantly, we require only that Q_K admit a tractable density, not the marginal distribution over y.

In SMCP^3, users specify particle updates by defining a *pair* of probabilistic computations:

Definition 2 (SMCP^3 move)**.** An SMCP^3 move from $\tilde{P}_{t-1}$ to $\tilde{P}_t$ is a pair of probabilistic computations $K_t = (U_{K_t}; Q_{K_t}; f_{K_t}) : X_{t-1} \quad X_t \quad U_{L_t}$ and $L_t = (U_{L_t}; Q_{L_t}; f_{L_t}) : X_t \quad X_{t-1} \quad U_{K_t}$, such that

- $\tilde{P}_t \quad Q_{L_t}$ is absolutely continuous with respect to $(\tilde{P}_{t-1} \quad Q_{K_t}) \quad f_{K_t}^{-1}$, and[3]
- on the support of $\tilde{P}_{t-1} \quad Q_{K_t}$, $f_{K_t} = f_{L_t}^{-1}$.

The K computation. K_t's role in Alg. 1 is to transform a value x^{t-1} from the particle approximation of $\tilde{P}_{t-1}$ into a value x^t for the particle approximation of $\tilde{P}_t$. To do so, it samples an auxiliary variable $u_{K_t} \ 2 \ U_{K_t}$ (L15) and then applies a measurable map f_K to $(x^{t-1}; u_{K_t})$ (L17). Computing an importance weight for this proposal would traditionally require intractable marginalization over auxiliary randomness in u_{K_t}, and over parts of x^{t-1} not retained in the update. To avoid this intractable integral, K_t also returns an auxiliary variable u_{L_t} (L17). Formally, the role of u_{L_t} is to ensure that f_{K_t}'s restriction to $\tilde{P}_{t-1} \quad Q_{K_t}$'s support is injective. Intuitively, u_{L_t} records information about $(x^{t-1}; u_{K_t})$ "lost" during the update. The "information loss" can arise for two reasons:

1. The information in the previous state x^{t-1} may not all be propagated to the next state. For example, a K_t that implements an MCMC-like update may overwrite components about the previous state. Then U_{L_t} can store the part of the old state x^{t-1} that was overwritten.
2. There may be several paths through the computation K_t that yield the same proposed x^t, so the auxiliary randomness u_{K_t} is impossible to recover exactly after the update. In this case, U_{L_t} can be a space parameterizing the *ways* that a particular output x^t could be generated from an input x^{t-1}.

The L computation. Similar to L kernels in Del Moral et al. (14), the L_t computation's overall goal is to "guess" how K_t generated a given x^t; in the case of SMCP^3, this means outputting a hypothesized previous particle x^{t-1}, and hypothesized auxiliary randomness u_{K_t}. It does this by sampling randomness $u_{L_t} \quad Q_{L_t}(x^t \ ! \ )$ and computing $(x^{t-1}; u_{K_t}) = f_{L_t}(x^t; u_{L_t})$. Since f_{L_t} and f_{K_t} are inverses, given K_t, Q_{L_t} is the only remaining degree of freedom; Proposition 1 gives a local optimality condition for this choice. Note that Alg. 1 never needs to run L_t; L_t is only used to compute importance weights.

[3] Following Kallenberg (26), we overload $P \quad Q$ to denote either the product of two measures, or the product of a measure P over X with a *kernel* $Q : X \ ! \ U$, to obtain a measure on the product space $X \quad U$. For measurable $A \quad X \quad U$, $(P \quad Q)(A) = \int\int_A P(dx)Q(x; du)$ Furthermore, we use the standard notation $\quad f^{\ 1}$ to denote the pushforward of $\quad$ by f, whether $\quad$ is a measure or a kernel.

Computing weights. Given an SMCP³ move, we can use K_t to propose a new particle value x^t (L14-17), but we still need a way of computing a new weight w^t so that $(x^t; w^t)$ is *properly weighted* for $\tilde{P}_t$.

Definition 3 (properly weighted (47))**.** A random variable $(x; w)$ taking values in X $\mathbb{R}_{\geq 0}$ is *properly weighted* for a measure $\tilde{P}$ over X if for all measurable $f : X \to \mathbb{R}$, $\mathbb{E}[w f(x)] = \int_X f(x)\tilde{P}(dx)$.

The following justifies Alg. 1's weight computation:

Theorem 1. Let $(K; L)$ be an SMCP³ move from $\tilde{P}_{t-1}$ to $\tilde{P}_t$. If $(x; w)$ is properly weighted for $\tilde{P}_{t-1}$, then letting u_K $Q_K(x \to)$, and $(x'; u_L) = f_K(x; u_K)$, the pair $(x'; \hat{w}\ w)$ is properly weighted for $\tilde{P}_t$, where

$$\hat{w} = \frac{\tilde{p}_t(x')q_L(x' \to u_L)}{\tilde{p}_{t-1}(x)q_K(x \to u_K)} \frac{d(\ _t \ \ _L)}{d((\ _{t-1} \ \ _K)\ f_K^{-1})}(x'; u_L):$$

The first factor is a density ratio, and the second is a *change-of-variables* correction for the bijection f_K. When the spaces X_t, X_{t-1}, U_K and U_L are Euclidean with Lebesgue reference measures , and f_K is differentiable, the correction is the absolute value of f_K's Jacobian determinant. Correction factors can also be computed in more general settings (52), including when f_K can be expressed as a program in a Turing-complete language with piecewise-differentiable primitives (31). In Sec. 3, we show how to compute the correction when models and moves are defined as probabilistic programs in the Gen language (10).

Locally optimal L kernels. The choice of U_{L_t} is often straightforward, but the kernel Q_{L_t} may be less so. The following gives qualitative guidance:

Proposition 1. Let R_t be the measure $(\tilde{P}_{t-1}\ Q_{K_t})$ f_K^{-1} over X_t U_{L_t}. If $R_t = (R_t \ ^{-1}_1)$ Q_{L_t}, where $_1$ is the projection map onto the first coordinate (here, X_t), then Q_{L_t} is *locally optimal*: among all choices of Q_{L_t}, it minimizes the variance of the incremental weight $\hat{w}$.[4]

In other words, the locally optimal choice of Q_{L_t} attempts to recover the lost information U_{L_t} according to its conditional distribution given x^t. This choice of Q_{L_t} makes the incremental weight depend only on x^t; not the auxiliary randomness: $\hat{w} = \frac{d\tilde{P}_t}{d(R_t \circ \pi_1\ ^1)}(x^t)$.

Convergence of SMCP³. SMC algorithms that use SMCP³ updates can be formulated as Feynman-Kac models (13) (see supplement), so we can use standard arguments (6) to reason about their convergence.

[4]The optimality is only *local* in that, by incorporating knowledge about the specific update kernels applied at steps 1 through t 1, it is possible to design Q_{L_t} kernels that —although they yield higher-variance *incremental* weights—reduce the overall variance of $w^t = \hat{w}\ W^{t\ 1}$.

Proposition 2 (Central Limit Theorem)**.** If $\tilde{P}_t$, K_t, L_t are such that Alg. 1's incremental weights $\hat{w}_i^t$ are bounded above, then for any continuous, bounded function $' : X_t \to \mathbb{R}$, there exists s.t.

$$\sqrt{N}\left(\frac{1}{N}\sum_{n=1}^{N} w_n^t\ '(x_n^t) \quad \int_{X_t} '(x)\tilde{P}_t(dx)\right) \xrightarrow{D} N(0; \ )$$

as $N \to 1$, where $\xrightarrow{D}$ is convergence in distribution.

3 Automating SMCP³

The previous section presented a mathematical framework for designing SMC algorithms with particle updates based on probabilistic computations. In this section, we show how to *automate* correct implementations of SMCP³ algorithms, when the target distributions and particle updates are specified as source code in a probabilistic programming language.

Gen programs. We work with the PPL Gen (10), in which a probabilistic program is an ordinary (deterministic) Julia function, augmented with the syntactic construct `fname~ distribution`. This statement causes Gen to sample a random value from a distribution (e.g., `normal`$(0; 4)$), and to associate the value internally with the name `name`. When executing a Gen program, we can turn on *tracing* to obtain two outputs: the final value returned by the program, as well as a *trace*, recording the names and values of all encountered random variables. For example, when run with argument $t = 2$, the Gen program `model` in Fig. 2a might generate the trace `Trace("x1" )` 0:4;`"y1" )` 1:1;`"x2" )` 0:2;`"y2" )` 0:3). It is clear that the *values* in the trace will vary from run to run, but because programs may make control flow decisions on the basis of their samples, even the *number* of sampling statements encountered may vary (and with it, the set of *variable names* included in the trace).

The measure space of traces. The set of execution traces that Gen programs can generate form a measurable space T of probabilistic program traces. Each trace 2 T is given by a list of key-value pairs, associating the names of random variables (strings) with the values they assumed in a particular execution. In the supplement, we define a reference measure $_{\mathsf{T}}$ over T, with respect to which every Gen program's sampling distribution is absolutely continuous, enabling rigorous reasoning about the densities of these distributions.

Gen programs as probabilistic computations. A Gen probabilistic program taking arguments in space A and returning values in space B implements a probabilistic computation $P = (U_P; Q_P; f_P) : A \to B$. $Q_P(a \to)$ is the distribution over traces induced by the P, $U_P = \mathsf{T}$ is the space of traces, and $f_P : A \ \mathsf{T} \to$

```
@gen function model(t)
  x = 0
  for step=1:t
    x = {"x$(step)"} ~ normal(x, 1.0)
    y = {"y$(step)"} ~ normal(x, 1.0)
  end
end
```

(a) Gen implementation of a simple dynamic model.

```
function smc_move(t, new_obs)
  @gen function K(tr)
    # Sample u from dynamics model
    prev_x = tr["x$(t-1)"]
    u = {"u"} ~ normal(prev_x, 1.0)
    # Unadjusted Langevin move to sample an x
    x = {"x"} ~ ULA(u, prev_x, new_obs)
    # Update trace with newly proposed x
    tr["x$(t)"] = x
    # Return proposed trace & aux. randomness
    return (tr, Trace("u" => u))
  end
  @gen function L(tr)
    # Guess the aux. var that K sampled
    prev_x = tr["x$(t-1)"]
    u = {"u"} ~ normal(prev_x, 1.0)
    # Return previous time step's trace, and
    # trace of K that would bring us to this
    # trace.
    return (Trace(["x$(i)" => tr["x$(i)"]
                for i in 1:t-1]...),
            Trace("u" => u, "x" => tr["x$(t)"]))
  end
  return (K, L)
end
```

(b) An SMCP^3 move, specified using Gen programs.

Figure 2: Code for Section 4.1's SMCP^3 algorithm.

B is a function which computes the return value of the program, given its inputs and the random choices it made during execution (as captured in its trace). For any Gen program, Gen automates the implementation of several useful operations, including:

1. SAMPLE-TRACE(P, a): Sample $\tau \sim Q_P(a \to \cdot)$.
2. EVALUATE-LOGPDF(P, a, τ): Evaluate $\log q_P(a \to \tau) = \log \frac{d(Q_P(a \to \cdot))}{d\mu_{\mathbb{T}}}(\tau)$.
3. COMPUTE-RETVAL(P, a, τ): Compute $f_P(a, \tau)$.

We review the implementations of these operations (which are standard in PPLs) in the supplement.

Defining sequences of unnormalized measures. Given an argument a, a probabilistic program P defines a normalized probability distribution over traces containing many variables. Just as a joint probability density $p(x, y)$ over a pair can be recast as an *unnormalized* density $\tilde{p}(x) = p(x, y)$ for a fixed observation $\mathbf{y}$, we can specify unnormalized measures in Gen by fixing a *trace* $\mathbf{y} \in \mathbb{T}$ of observations, mapping the names of some of the variables that P samples to observed values. In particular, we define $\tilde{P}^{\mathbf{y}}$ as the unnormalized kernel with density $\tilde{p}^{\mathbf{y}}(a \to \tau) = q_P(a \to \tau \oplus \mathbf{y})$ w.r.t. $\mu_{\mathbb{T}}$, where $\oplus$ *merges* two traces with disjoint sets of names. (If τ shares any names with $\mathbf{y}$, we define $\tilde{p}^{\mathbf{y}}(\tau)$ to be 0.) A user defines a *sequence* of unnormalized distributions via (1) a probabilistic program P with argument space $A = \{1, \ldots, T\}$ (the argument gives the position in the sequence), and (2) a sequence $(\mathbf{y}_t)_{t=1}^{T}$ of observation traces for each time. This yields sequence $(\tilde{P}^{\mathbf{y}_t}(t \to \cdot))_{t=1}^{T}$.

Algorithm 2 Automated SMCP^3 update

Require: model Gen program P
Require: previous and current observations y_{t-1}, y_t
Require: (x, w) properly wtd. for $\tilde{P}^{y_{t-1}}(t-1 \to \cdot)$
Require: Gen programs K_t, L_t specifying move
Ensure: (x', w') properly weighted for $\tilde{P}^{y_t}(t \to \cdot)$

1: $u_{K_t} \leftarrow$ SAMPLE-TRACE(K_t, x)
2: $((x', u_{L_t}), \hat{J}) \leftarrow$ AD(COMPUTE-RETVAL(K_t, x, u_{K_t}))
3: $\log q_{K_t} \leftarrow$ EVALUATE-LOGPDF(K_t, x, u_{K_t})
4: $\log \tilde{p}_{t-1} \leftarrow$ EVALUATE-LOGPDF$(P, t-1, x \oplus y_{t-1})$
5: $\log \tilde{p}_t \leftarrow$ EVALUATE-LOGPDF$(P, t, x' \oplus y_t)$
6: $\log q_{L_t} \leftarrow$ EVALUATE-LOGPDF(L_t, x', u_{L_t})
7: $\log r \leftarrow \log \tilde{p}_t - \log \tilde{p}_{t-1} + \log q_{L_t} - \log q_{K_t}$
8: $\log w' \leftarrow \log w + \log r + \log |\det \hat{J}|$
9: **return** $(x', \log w')$

Specifying K and L. For each $t \in \{2, \ldots, T\}$, the user can specify $K_t : \mathbb{T} \to \mathbb{T} \times \mathbb{T}$ and $L_t : \mathbb{T} \to \mathbb{T} \times \mathbb{T}$, as probabilistic programs that accept traces as input and return pairs of traces as output. See Fig. 2b for an example. (Since both the models and proposals are defined as probabilistic programs, the state spaces X_t and the auxiliary spaces U_K, U_L from Sec. 2 are all $\mathbb{T}$.) The condition that f_{K_t} and f_{L_t} can be probabilistically fuzz-tested on automatically generated input traces.

Automating SMCP^3. Given model probabilistic program P, observations $\mathbf{y}_{t-1}$ and $\mathbf{y}_t$, and K_t and L_t, Algorithm 2 automates the SMCP^3 update from a particle (x, w) properly weighted for $\tilde{P}_{t-1} = \tilde{P}^{\mathbf{y}_{t-1}}(t-1 \to \cdot)$ to a particle (x', w') properly weighted for $\tilde{P}_t = \tilde{P}^{\mathbf{y}_t}(t \to \cdot)$. It samples $u_K \sim Q_{K_t}(x \to \cdot)$ (L1), computes $(x', u_L) = f_{K_t}(x, u_K)$ (L2), evaluates model and proposal densities (L3-6),[5] and finally computes the particle weight update (L7-8). To compute the weight update $\hat{w}$ from Theorem 1, step (4) must compute the change-of-measures correction $\frac{d(\mu_{\mathbb{T}_t} \otimes \mu_{\mathbb{T}})}{d((\mu_{\mathbb{T}_{t-1}} \otimes \mu_{\mathbb{T}}) \circ f_{K_t}^{-1})}(x', u_L)$, where $\mathbb{T}_t$ is the subset of $\mathbb{T}$ supported by $\tilde{P}_t$, and $\mu_{\mathbb{T}_t}$ is the restriction of $\mu_{\mathbb{T}}$ to $\mathbb{T}_t$. Theorem 2 gives a concrete expression for this factor.

Theorem 2. Let $\mathbb{T}_1^2$ and $\mathbb{T}_2^2$ be measurable subsets of $\mathbb{T} \times \mathbb{T}$, and let μ_1 and μ_2 be restrictions of $\mu_{\mathbb{T}} \otimes \mu_{\mathbb{T}}$

[5]On the cost of Alg. 2 computing 4 trace densities: Gen features optimizations that exploit cancellations in density ratios to achieve asymptotic speedups where possible (10).

to these subsets. Suppose $f : \mathsf{T}_1^2 \to \mathsf{T}_2^2$ is a PAP bijection[6] and that $\__2$ is absolutely continuous with respect to $\__1 \circ f^{-1}$. Then

$$\frac{d\ \__2}{d(\__1 \circ f^{-1})}(f(\__1; \__2)) = |\det(J f_{\tau_1,\tau_2})(\_(\__1) +\!\!+ \_(\__2))|;$$

where $\_ : \mathsf{T} \to \bigcup_{d\in\mathbb{N}} \mathbb{R}^d$ extracts all the real-valued entries in a trace into a vector, $\__\tau : \mathbb{R}^{|\rho(\tau)|} \to \mathsf{T}$ replaces the real entries in $\_$ with values from a vector, and $f_{\tau_1,\tau_2} : \mathbb{R}^{|\rho(\tau_1)|+|\rho(\tau_2)|} \to \mathbb{R}^{|\rho(\tau_1)|+|\rho(\tau_2)|}$ accepts as input the concatenation of $\mathbf{v}_1 \in \mathbb{R}^{|\rho(\tau_1)|}$ and $\mathbf{v}_2 \in \mathbb{R}^{|\rho(\tau_2)|}$, computes $\mathbf{v}_1'; \mathbf{v}_2' = f(\__{\tau_1}(\mathbf{v}_1); \__{\tau_2}(\mathbf{v}_2))$, and then returns the vector concatenation $\_(\mathbf{v}_1') +\!\!+ \_(\mathbf{v}_2')$.

Line 2 of Algorithm 2 uses AD to to compute the Jacobian $\hat{J} = J f_{K_t x, u_{K_t}}$; per Theorem 2, $|\det \hat{J}|$ is the change-of-measure term needed to compute $\hat{w}$.[7]

4 Examples

The examples in this section demonstrate, in multiple domains, that probabilistic program proposals can improve inference relative to strong SMC baselines that use proposals with tractable densities. They also demonstrate that resample-move SMC algorithms can often be improved by moving complex proposal logic out of MCMC rejuvenation steps, into SMCP^3 moves.

4.1 Online inference in state-space models

Model. We first study the 1D linear Gaussian dynamic model from Fig. 2a. We take $x_0 = 0$, and for $t > 0$, set $x_t \sim \mathcal{N}(x_{t-1}; 1)$ and $y_t \sim \mathcal{N}(x_t; 1)$. The t^{th} target is the filtering posterior $p_t(x_{1:t} \mid y_{1:t})$.

SMCP^3 algorithm. We set K_t and L_t as in Fig. 2b. As illustrated in Fig. 4, K_t extends $x_{1:t-1}$ with a new value x_t, generated by performing an unadjusted Langevin ascent move from a random initial position $u \sim \mathcal{N}(x_{t-1}; 1)$. Our simple choice of L_t ignores x_t when proposing u, generating it from $\mathcal{N}(x_{t-1}; 1)$.

Baselines. We compare to two baselines: a standard bootstrap particle filter, which proposes x_t from its prior, and a resample-move SMC algorithm, which additionally runs Metropolis-adjusted Langevin ascent (MALA) rejuvenation on x_t after each step.

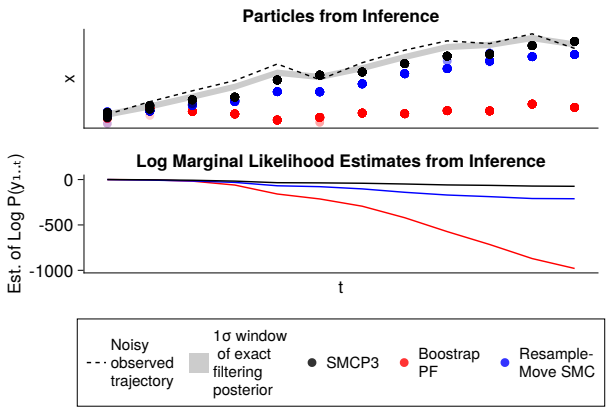


Figure 3: Example particles and log-marginal-likelihood estimates over time from online state estimation using the algorithms from Section 4.1.

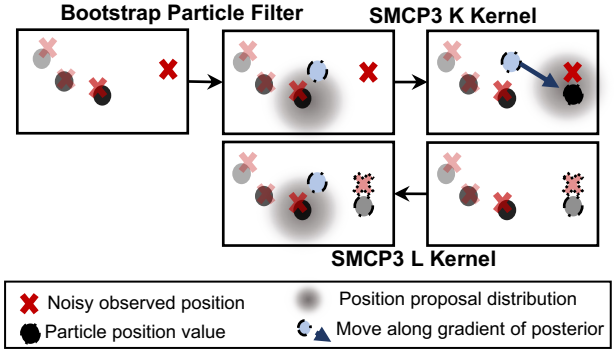
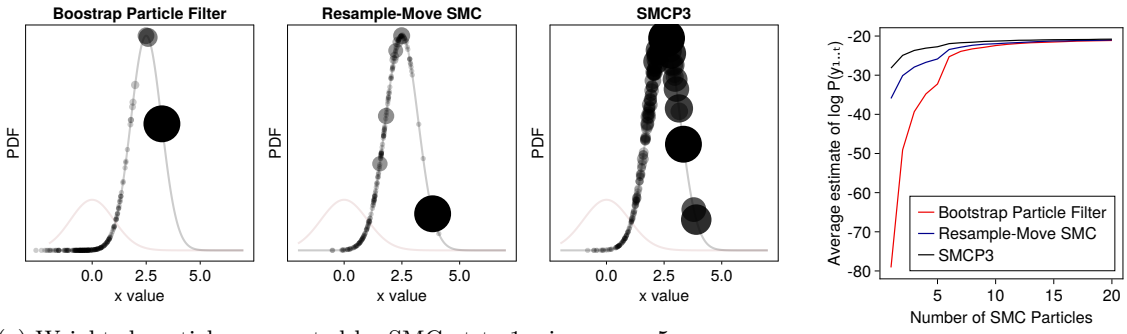


Figure 4: 2D variant of the SMCP^3 kernels from Figure 2. **Top left**: at $t = 4$, a new noisy position is observed. **Top middle**: the bootstrap particle filter proposes latent position z_t (blue circle) from the dynamics model given z_{t-1}. **Top right**: the SMCP^3 kernel improves on this proposal by moving along $r \log P(z_t \mid y_t; z_{t-1})$ (arrow), and sampling z_t' (black circle) nearby. **Bottom**: L proposes z_t given z_t'.

Results. Fig. 3 illustrates inference on one example sequence of observations, and Fig. 5b plots log marginal likelihood estimates, averaged over many synthetic datasets, for varying numbers of particles. Our algorithm significantly outperforms the bootstrap filter, and consistently yields better results than the stronger resample-move baseline. Figure 5a sheds light on why this is. The bootstrap particle filter proposes from the prior, and lands only a few particles near the mode of the posterior. The resample-move algorithm's MALA rejuvenation successfully moves these proposals closer to the mode, but does not update particle weights to reflect this progress, so promising particles may be lost during resampling. SMCP^3 also uses Langevin ascent to move the particles, but accounts for

[6] PAP stands for *piecewise analytic under analytic partition* (29), which can be defined not only in the Euclidean setting but also more generally (31), including for T. Every function implementable in a programming language with PAP primitives, if statements, and recursion is PAP (31).

[7] Our implementation uses forward-mode AD (54): it replaces every real number in the input traces x and u_{K_t} with dual numbers, runs `COMPUTE-RETVAL` (i.e., f_{K_t}), and then reads out the dual numbers that end up stored in the returned traces $x^0; u_{L_t}$, to fill out the Jacobian matrix $\hat{J}$. (We delete the dual components of the dual numbers before applying further operations to x^0 and u_{L_t} on L5-6.)



(a) Weighted particles generated by SMC at t=1, given $y_1 = 5$. Pink curve: PDF of $P(X_1)$. Grey curve: PDF of the target $P(X_1 | y_1)$.

(b) Inference performance.

Figure 5: SMCP^3 vs baselines for online state estimation, in the model from Figure 2a. In (c), inference performance is measured by the average log-marginal-likelihood estimate, $\log P(y_{1:10})$, produced by each SMC algorithm for 20 random 10-step observation sequences generated from the model.

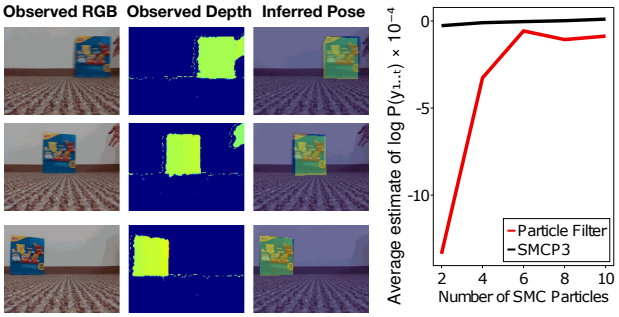


Figure 6: Object tracking from depth video in a non-differentiable state-space model.

this move in its weights, so that after resampling the particle cloud will better reflect the new posterior.

Non-differentiable variant. In Fig. 6, we also consider a variant of the model that uses a likelihood based on a non-differentiable renderer, for object tracking from depth video. Instead of performing a Langevin move from a randomly initialized point, the K proposal program samples a proposed point from a coarse grid centered at a random location, based on relative likelihoods of all the grid's points. As in the toy differentiable model, this improves over the bootstrap particle filter baseline; see supplement for details.

4.2 Online inference in mixture models

Model. We next consider a sequence of collapsed Dirichlet process mixture models (DPMMs), each incorporating one additional datapoint. The t^{th} model places a $CRP(f1;:::;tg)$ prior over partitions Π_t of $f1;:::;tg$, and for each cluster C 2 Π_t, generates data $(y_i)_{i \in C}$ jointly from an exchangeable likelihood $F(\mathbf{y})$.[8] We observe data $y_{1:t}$ and wish to infer the partition Π_t.

[8] We use three different likelihoods F, to model synthetic Gaussian data, real-world astronomical data, and strings in a Medicare dataset; see supplement for details.

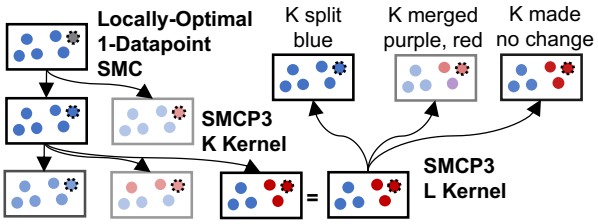


Figure 7: SMCP^3 kernels for data-clustering (Section 4.2). **Top**: new datapoint y_6 (grey) is added to a particle where datapoints $y_{1:5}$ form a single (blue) cluster. Locally-optimal SMC either chooses to (A) create a new (red) cluster containing y_6 (unlikely), or (B) merge y_6 into the blue cluster. In case (B), our SMCP^3 K kernel may then perform a cluster split move. **Bottom**: the SMCP^3 L kernel sees a clustering containing y_6 (pink), and must propose what clustering of $\mathbf{y}_{1:5}$ had existed before the K kernel incorporated y_6.

SMCP^3 algorithm. Fig. 7 illustrates our algorithm. K_t accepts as input a partition Π_{t-1} of the first t 1 datapoints and proposes a partition Π_t that incorporates the new datapoint y_t. To do so, it first performs a "Gibbs" assignment of y_t to an existing cluster C_* 2 Π_{t-1} or to a new cluster; if a new cluster, K_t stops early and proposes $\Pi_t := \Pi_{t-1}$ [$fftgg$. Otherwise, K_t decides between splitting C_*, merging C_* with another cluster, or leaving C_* as is. This choice is made based on an importance-sampling estimate of the total probability of all states in which C_* is split; see supplement for details. L_t identifies cluster C'_* 2 Π_t containing t, then chooses to either split, merge, or not change it to recover C_*.

Baselines. We compare to two baselines: an SMC algorithm that uses the locally optimal "Gibbs" proposal to incorporate y_t into the clustering (without splitting or merging), and a resample-move extension which additionally applies split/merge rejuvenation.

	Est. of $\log P(\mathbf{y}_{1:T})$	
Galaxy data (random order)		
Locally Optimal SMC	-421.97	1.13
Resample-Move Split/Merge	-422.00	0.72
SMCP^3 Split/Merge	-422.28	0.96
Galaxy data (sorted)		
Locally Optimal SMC	-425.78	1.75
Resample-Move Split/Merge	**-422.07**	**1.03**
SMCP^3 Split/Merge	**-422.18**	**1.04**
Medicare data		
Locally Optimal SMC	-40383.00	688.77
Resample-Move Split/Merge	-14233.61	53.10
SMCP^3 Split/Merge	**-13882.10**	**0.27**

Table 1: Results for the mixture model, reporting the estimate each algorithm returns of the log-marginal-likelihood of a dataset $\mathbf{y}_{1:T}$; higher is better.

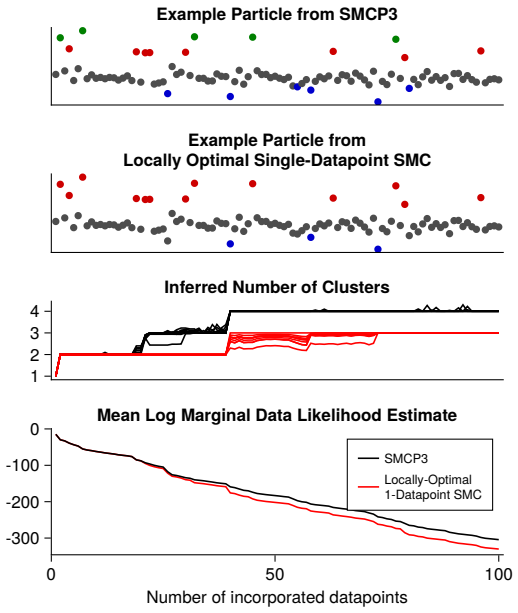


Figure 8: Clustering via locally optimal SMC, and SMCP^3, on an illustrative synthetic dataset. **Panels 1, 2**: example particles showing clusterings $\mathbf{y}_{1:T}$ from each algorithm; each point $(x; y)$ means datapoint $\mathbf{y}_x = y$. **Panel 3**: mean number of inferred clusters from 10 runs of each algorithm, over time. SMCP^3 consistently finds all 4 clusters likely under the posterior; the baseline only finds 3. **Panel 4**: poor inference is reflected in lower estimated $\log P(\mathbf{y}_{1:T})$.

Results. Table 1 shows estimates of the log marginal likelihood produced by each algorithm on two real datasets. As Fig. 8 illustrates, the locally optimal baseline's greedy assignment of points to clusters can lead it to get stuck in local modes, a phenomenon we also observe in the Medicare dataset (40) and in pathological data *orderings* on the Galaxy dataset (8, 18). Both SMCP^3 and resample-move are able to escape local modes, but as in Sec. 4.1, we see evidence on the Medicare and synthetic datasets that SMCP^3 computes better weights after splits or merges, leading to improved log likelihood estimates.

5 Related Work and Discussion

SMC. SMCP^3 provides automation for implementors of a broad variety of SMC algorithms, including SMC samplers (14), resample-move SMC (24), and move-reweight SMC (39). Unlike this prior work, SMCP^3 can be used to compute proper SMC weights for proposals that incorporate auxiliary variables (21, 22, 32) and deterministic transformations. Many algorithms use SMC moves as building blocks, including extensions to SMC (5, 27, 34) and pseudomarginal algorithms (2, 33); SMCP^3 moves could be incorporated into these algorithms. SMCP^3 has broad coverage, but does not cover techniques that *adapt* targets or proposals based on the current particle cloud (3, 20), which may compromise proper weighting.

Probabilistic programming. Many PPLs support automated SMC for models specified as probabilistic programs (10, 23, 35, 38, 41, 50, 56, 58, 59). Some also have support for restricted classes of custom SMC with proposals defined as probabilistic programs (4, 10, 43, 45, 57). These languages' restrictions prohibit proposal programs that sample auxiliary variables or propose deterministic transformations of samples. Stites et al. (57) present a set of combinators for defining samplers, including a `propose` combinator for specifying custom SMC proposals. Although these proposals can use auxiliary variables, they are ignored for weight computation, yielding sound but high-variance weights (equivalent to using a particular sub-optimal choice of L kernel in SMCP^3). SMCP^3 is inspired by automated involutive MCMC (9), an analogously general framework for automated, custom MH with probabilistic program proposals.

Discussion. SMCP^3 gives inference algorithm designers a more flexible framework than previous formulations, and also automates implementation details. Initial experiments show that custom SMCP^3 proposals using the new degrees of freedom it offers can yield more accurate inference than strong baselines. One important area for future research is to improve speed by applying PPL compilation techniques (10, 25, 36, 44, 50, 60) and by leveraging massively parallel hardware (19, 36, 46, 51). It also may be possible to automatically tune the runtime and robustness of probabilistic program proposals, using data-dependent (8) and model-averaged (15) estimators of SMC inference accuracy, or recently introduced methods for differentiating through SMC (7, 28, 37, 48, 55, 61). We hope SMCP^3 helps practitioners apply advanced SMC to complex modeling problems. We also hope it encourages experts to experiment with proposals that

leverage richer probabilistic computations, for example via Markov kernels based on backtracking search, optimization, and model-based planning algorithms.

References

[1] Christian Andersson Naesseth, Fredrik Lindsten, and Thomas B Schön. Sequential monte carlo for graphical models. *Advances in Neural Information Processing Systems*, 27, 2014.

[2] Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. Particle markov chain monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342, 2010.

[3] Alexandros Beskos, Ajay Jasra, Nikolas Kantas, and Alexandre Thiery. On the convergence of adaptive sequential monte carlo methods. *The Annals of Applied Probability*, 26(2):1111–1146, 2016.

[4] Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20 (1):973–978, 2019.

[5] Nicolas Chopin, Pierre E Jacob, and Omiros Papaspiliopoulos. Smc2: an efficient algorithm for sequential analysis of state space models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 75(3):397–426, 2013.

[6] Nicolas Chopin, Omiros Papaspiliopoulos, et al. *An introduction to sequential Monte Carlo*. Springer, 2020.

[7] Adrien Corenflos, James Thornton, George Deligiannidis, and Arnaud Doucet. Differentiable particle filtering via entropy-regularized optimal transport. In *International Conference on Machine Learning*, pages 2100–2111. PMLR, 2021.

[8] Marco Cusumano-Towner and Vikash K Mansinghka. Aide: An algorithm for measuring the accuracy of probabilistic inference algorithms. *Advances in Neural Information Processing Systems*, 30, 2017.

[9] Marco Cusumano-Towner, Alexander K Lew, and Vikash K Mansinghka. Automating involutive mcmc using probabilistic and differentiable programming. *arXiv preprint arXiv:2007.09871*, 2020.

[10] Marco F Cusumano-Towner, Feras A Saad, Alexander K Lew, and Vikash K Mansinghka. Gen: a general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th acm sigplan conference on programming language design and implementation*, pages 221–236, 2019.

[11] Chenguang Dai, Jeremy Heng, Pierre E Jacob, and Nick Whiteley. An invitation to sequential monte carlo samplers. *Journal of the American Statistical Association*, (just-accepted):1–38, 2022.

[12] P Del Moral, A Doucet, and A Jasra. Sequential monte carlo for bayesian computation. *Bayesian statistics*, 8, 2006.

[13] Pierre Del Moral. *Feynman-Kac formulae: genealogical and interacting particle systems with applications*, volume 88. Springer, 2004.

[14] Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential monte carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(3):411–436, 2006.

[15] Justin Domke. An easy to interpret diagnostic for approximate inference: Symmetric divergence over simulations. *arXiv preprint arXiv:2103.01030*, 2021.

[16] Randal Douc and Olivier Cappé. Comparison of resampling schemes for particle filtering. In *Ispa 2005. proceedings of the 4th international symposium on image and signal processing and analysis, 2005.*, pages 64–69. IEEE, 2005.

[17] Arnaud Doucet, Nando De Freitas, Neil James Gordon, et al. *Sequential Monte Carlo methods in practice*, volume 1. Springer, 2001.

[18] Michael J Drinkwater, Quentin A Parker, Dominique Proust, Eric Slezak, and Hernán Quintana. The large scale distribution of galaxies in the shapley supercluster. *Publications of the Astronomical Society of Australia*, 21(1):89–96, 2004.

[19] Garland Durham and John Geweke. Massively parallel sequential monte carlo for bayesian inference. *Manuscript, URL http://www. censoc. uts. edu. au/pdfs/geweke_papers/gp_ working_9. pdf. Nalan Basturk, Lennart Hoogerheide, Anne Opschoor, Herman K. van Dijk*, 29:17–34, 2011.

[20] Paul Fearnhead and Benjamin M Taylor. An adaptive sequential monte carlo sampler. *Bayesian analysis*, 8(2):411–438, 2013.

[21] Paul Fearnhead, Omiros Papaspiliopoulos, Gareth O Roberts, and Andrew Stuart. Random-weight particle filtering of continuous time processes. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(4): 497–512, 2010.

[22] Axel Finke. *On extended state-space constructions for Monte Carlo methods*. PhD thesis, University of Warwick, 2015.

[23] Hong Ge, Kai Xu, and Zoubin Ghahramani. Turing: a language for flexible probabilistic inference. In *International conference on arti cial intelligence and statistics*, pages 1682–1690. PMLR, 2018.

[24] Walter R Gilks and Carlo Berzuini. Following a moving target—monte carlo inference for dynamic bayesian models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(1):127–146, 2001.

[25] Daniel Huang, Jean-Baptiste Tristan, and Greg Morrisett. Compiling markov chain monte carlo algorithms for probabilistic modeling. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 111–125, 2017.

[26] Olav Kallenberg. *Foundations of modern probability*, volume 2. Springer, 1997.

[27] Juan Kuntz, Francesca R Crucinio, and Adam M Johansen. The divide-and-conquer sequential monte carlo algorithm: theoretical properties and limit theorems. *arXiv preprint arXiv:2110.15782*, 2021.

[28] Jinlin Lai, Justin Domke, and Daniel Sheldon. Variational marginal particle filters. In *International Conference on Arti cial Intelligence and Statistics*, pages 875–895. PMLR, 2022.

[29] Wonyeol Lee, Hangyeol Yu, Xavier Rival, and Hongseok Yang. On correctness of automatic differentiation for non-differentiable functions. *Advances in Neural Information Processing Systems*, 33:6719–6730, 2020.

[30] Alexander Lew, Monica Agrawal, David Sontag, and Vikash Mansinghka. Pclean: Bayesian data cleaning at scale with domain-specific probabilistic programming. In *International Conference on Arti cial Intelligence and Statistics*, pages 1927–1935. PMLR, 2021.

[31] Alexander K Lew, Mathieu Huot, and Vikash K Mansinghka. Towards denotational semantics of ad for higher-order, recursive, probabilistic languages. *arXiv preprint arXiv:2111.15456*, 2021.

[32] Alexander K. Lew, Marco Cusumano-Towner, and Vikash Mansinghka. Recursive monte carlo and variational inference with auxiliary variables. In *The 38th Conference on Uncertainty in Arti cial Intelligence*, 2022. URL https://openreview.net/forum?id=BzMxEdIsqeq.

[33] Fredrik Lindsten, Michael I Jordan, and Thomas B Schon. Particle gibbs with ancestor sampling. *Journal of Machine Learning Research*, 15:2145–2184, 2014.

[34] Fredrik Lindsten, Adam M Johansen, Christian A Naesseth, Bonnie Kirkpatrick, Thomas B Schön, JAD Aston, and Alexandre Bouchard-Côté. Divide-and-conquer with sequential monte carlo. *Journal of Computational and Graphical Statistics*, 26(2):445–458, 2017.

[35] Daniel Lundén, Johannes Borgström, and David Broman. Correctness of sequential monte carlo inference for probabilistic programming languages. In *ESOP*, pages 404–431, 2021.

[36] Daniel Lundén, Joey Öhman, Jan Kudlicka, Viktor Senderov, Fredrik Ronquist, and David Broman. Compiling universal probabilistic programming languages with efficient parallel sequential monte carlo inference. In *ESOP*, pages 29–56, 2022.

[37] Chris J Maddison, John Lawson, George Tucker, Nicolas Heess, Mohammad Norouzi, Andriy Mnih, Arnaud Doucet, and Yee Teh. Filtering variational objectives. *Advances in Neural Information Processing Systems*, 30, 2017.

[38] Vikash K Mansinghka, Ulrich Schaechtle, Shivam Handa, Alexey Radul, Yutian Chen, and Martin Rinard. Probabilistic programming with programmable inference. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 603–616, 2018.

[39] Reinaldo A Gomes Marques and Geir Storvik. Particle move-reweighting strategies for online inference. *Preprint series. Statistical Research Report http://urn. nb. no/URN: NBN: no-23420*, 2013.

[40] Medicare. Hospital compare, 2012.

[41] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L Ong, and Andrey Kolobov. Blog: Probabilistic models with unknown objects. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2006.

[42] Jeffrey Miller, Brenda Betancourt, Abbas Zaidi, Hanna Wallach, and Rebecca C Steorts. Microclustering: When the cluster sizes grow sublinearly with the size of the data set. *arXiv preprint arXiv:1512.00792*, 2015.

[43] Lawrence M Murray. Bayesian state-space modelling on high-performance hardware using libbi. *arXiv preprint arXiv:1306.3277*, 2013.

[44] Lawrence M Murray. Lazy object copy as a platform for population-based probabilistic programming. *arXiv preprint arXiv:2001.05293*, 2020.

[45] Lawrence M Murray and Thomas B Schön. Automated learning with a probabilistic programming language: Birch. *Annual Reviews in Control*, 46: 29–43, 2018.

[46] Lawrence M Murray, Anthony Lee, and Pierre E Jacob. Parallel resampling in the particle filter. *Journal of Computational and Graphical Statistics*, 25(3):789–805, 2016.

[47] Christian Naesseth, Fredrik Lindsten, and Thomas Schon. Nested sequential monte carlo methods. In *International Conference on Machine Learning*, pages 1292–1301. PMLR, 2015.

[48] Christian Naesseth, Scott Linderman, Rajesh Ranganath, and David Blei. Variational sequential monte carlo. In *International conference on arti cial intelligence and statistics*, pages 968–977. PMLR, 2018.

[49] Christian A Naesseth, Fredrik Lindsten, Thomas B Schön, et al. Elements of sequential monte carlo. *Foundations and Trends® in Machine Learning*, 12(3):307–392, 2019.

[50] Brooks Paige and Frank Wood. A compilation target for probabilistic programming languages. In *International Conference on Machine Learning*, pages 1935–1943. PMLR, 2014.

[51] Brooks Paige, Frank Wood, Arnaud Doucet, and Yee Whye Teh. Asynchronous anytime sequential monte carlo. *Advances in neural information processing systems*, 27, 2014.

[52] Alexey Radul and Boris Alexeev. The base measure problem and its solution. In *International Conference on Arti cial Intelligence and Statistics*, pages 3583–3591. PMLR, 2021.

[53] Carl Rasmussen. The infinite gaussian mixture model. *Advances in neural information processing systems*, 12, 1999.

[54] Jarrett Revels, Miles Lubin, and Theodore Papamarkou. Forward-mode automatic differentiation in julia. *arXiv preprint arXiv:1607.07892*, 2016.

[55] Adam Ścibior and Frank Wood. Differentiable particle filtering without modifying the forward pass. *arXiv preprint arXiv:2106.10314*, 2021.

[56] Adam Ścibior, Zoubin Ghahramani, and Andrew D Gordon. Practical probabilistic programming with monads. In *Proceedings of the 2015 ACM SIGPLAN Symposium on Haskell*, pages 165–176, 2015.

[57] Sam Stites, Heiko Zimmermann, Hao Wu, Eli Sennesh, and Jan-Willem van de Meent. Learning proposals for probabilistic programs with inference combinators. In *Uncertainty in Arti cial Intelligence*, pages 1056–1066. PMLR, 2021.

[58] David Tolpin, Jan-Willem van de Meent, Hongseok Yang, and Frank Wood. Design and implementation of probabilistic programming language anglican. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional programming Languages*, pages 1–12, 2016.

[59] Frank Wood, Jan Willem Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In *Arti cial intelligence and statistics*, pages 1024–1032. PMLR, 2014.

[60] Yi Wu, Lei Li, Stuart Russell, and Rastislav Bodik. Swift: Compiled inference for probabilistic programming languages. *arXiv preprint arXiv:1606.09242*, 2016.

[61] Michael Zhu, Kevin Murphy, and Rico Jonschkowski. Towards differentiable resampling. *arXiv preprint arXiv:2004.11938*, 2020.

Supplementary Material

This supplement contains:

- Omitted proofs (of Theorem 1, Theorem 2, and Proposition 2), in Appendices A, B, and D.
- A definition of the measurable space T of traces, and the reference measure $\quad_{\mathsf{T}}$ (Appendix C).
- A brief introduction to the standard techniques Gen uses to automate the `SAMPLE-TRACE`, `EVAL-LOGPDF`, and `COMPUTE-RETVAL` operations (Appendix E).
- Full details on experimental setup, for each experiment in Section 4 (Appendix F).

A Proof of Theorem 1

Definition 4. Let $\quad$, $\quad$ be measures on X. Then $R(\ ;\ )$, the *restriction of* $\quad$ *to the support of* $\quad$, is the measure mapping a set $E \subseteq X$ to $\inf_{A \in \mathcal{X}} f\ (E \cap A)\ j\ \quad(A) = 0 g$.

Remark 1. Note that $R(\ ;\ )$ is absolutely continuous with respect to $\quad$, and that if some other measure P is absolutely continuous with respect to both $\quad$ and $\quad$, it is absolutely continuous with respect to $R(\ ;\ )$, and $\frac{dP}{dR(\mu,\nu)} = \frac{dP}{d\mu}$.

Lemma 1. Let P and Q be probability measures on spaces X and Y respectively. Let $\tilde{Q} = Z_Q\ Q$ and $\tilde{P} = Z_P\ P$ denote unnormalized versions of these measures, with densities $\tilde{p}$ and $\tilde{q}$ with respect to reference measures $\quad_X$ and $\quad_Y$. Let $S = Q\quad f^{-1}$ for some measurable bijection $f : X\ !\ Y$, and suppose that $\tilde{P}$ is absolutely continuous with respect to S. Then $(y; w)$ is properly weighted for $\frac{Z_P}{Z_Q}P$, where $y = f(x)$, $x\quad Q$, and $w = \frac{\tilde{p}(y)}{\tilde{q}(x)}\ \frac{dR(\mu_X, S)}{d(R(\mu_X, Q) \circ f^{\ 1})}(y)$.

Proof. Let $x\quad Q$ and $y = f(x)$. We note:

$$
\begin{aligned}
\frac{dP}{dS}(y) &= \frac{dP}{dR(\ _Y; S)}(y)\ \ \frac{dR(\ _Y; S)}{dS}(y) && \text{(Radon-Nikodym chain rule)}\\
&= \frac{\tilde{p}(y)}{Z_P}\ \ \frac{dR(\ _Y; S)}{d(R(\ _X; Q)\quad f^{-1})}(y)\ \ \frac{d(R(\ _X; Q)\quad f^{-1})}{d(Q\quad f^{-1})}(y) && \text{(density of } P\text{, Radon-Nikodym chain rule)}\\
&= \frac{\tilde{p}(y)}{Z_P}\ \ \frac{dR(\ _Y; S)}{d(R(\ _X; Q)\quad f^{-1})}(y)\ \ \frac{dR(\ _X; Q)}{dQ}(f^{-1}(y)) && \text{(pushforward by measurable bijection)}\\
&= \frac{\tilde{p}(y)}{Z_P}\ \ \frac{dR(\ _Y; S)}{d(R(\ _X; Q)\quad f^{-1})}(y)\ \ \frac{Z_Q}{\tilde{q}(x)} && \text{(density of } Q\text{, } x = f^{-1}(y))\\
&= \frac{Z_Q}{Z_P}\ \ w: && \text{(definition of } w)
\end{aligned}
$$

On the first line, we use the fact that since P is absolutely continuous with respect to both $\quad_Y$ and S, it is absolutely continuous with respect to $R(\ _Y; S)$. On the second line, we use the fact that since $S\quad\ _X\quad f^{-1}$ and $S\quad Q\quad f^{-1}$, $S\quad(R(\ _X; Q)\quad f^{-1})$, and therefore $R(\ _Y; S)\quad S\quad(R(\ _X; Q)\quad f^{-1})$:

Multiplying by $\frac{Z_P}{Z_Q}$ on each side, we have that $w = \frac{Z_P}{Z_Q}\frac{dP}{dS}(y)$, so for any measurable $g : Y\ !\ \mathsf{R}_{\geq 0}$, we have $\mathsf{E}[wg(y)] = \int \frac{Z_P}{Z_Q}\frac{dP}{dS}(y) g(y) S(dy) = \int g(y)(\frac{Z_P}{Z_Q}P)(dy)$. This is precisely the criterion for $(y; w)$ to be properly weighted for $\frac{Z_P}{Z_Q}P$. □

Theorem 1. Let $(K; L)$ be an SMCP^3 move from $\tilde{P}_{t-1}$ to $\tilde{P}_t$. If $(x; w)$ is properly weighted for $\tilde{P}_{t-1}$, then letting $u_K\quad Q_K(x\ !\ \ )$, and $(x'; u_L) = f_K(x; u_K)$, the pair $(x'; \hat{w}\quad w)$ is properly weighted for $\tilde{P}_t$, where

$$
\hat{w} = \frac{\tilde{p}_t(x') q_L(x'\ !\ u_L)}{\tilde{p}_{t-1}(x) q_K(x\ !\ u_K)}\ \ \frac{dR(\ _t\quad\ _L; (\tilde{P}_{t-1}\quad Q_K)\quad f_K^{-1})}{d(R(\ _{t-1}\quad\ _K; \tilde{P}_{t-1}\quad Q_K)\quad f_K^{-1})}(x'; u_L):
$$

Proof. We apply Lemma 1, with $X := X_{t-1} \quad U_K$, $Y := X_t \quad U_{L_t}$, $f := f_K$, $\tilde{P} := \tilde{P}_t \quad Q_L$, and $\tilde{Q} := \tilde{P}_{t-1} \quad Q_K$. This tells us that for all measurable $g : X_t \quad U_L \; ! \; \mathsf{R}_{\geq 0}$, $\mathsf{E}_{x\sim P_t \;\; 1, u_K \sim Q_K}[\hat{w} \;\; g(f_K(x; u_K))] = \int \frac{Z_{P_t}}{Z_{P_t \;\; 1}} g(x'; u_L)(P_t \quad Q_L)(d(x'; u_L))$. In particular, for all $h : X_t \; ! \; \mathsf{R}_{\geq 0}$, we can take $g(x'; u_L) = h(x')$, and get that $\mathsf{E}_{x\sim P_t \;\; 1, u_K \sim Q_K}[\hat{w} \;\; h(\;\; 1(f_K(x; u_K)))] = \int \frac{Z_{P_t}}{Z_{P_t \;\; 1}} h(x') P_t(dx')$.

We now apply the assumption that $(x; w)$ is properly weighted for $\tilde{P}_{t-1} = Z_{P_t \;\; 1} P_{t-1}$ to rewrite the expectation w.r.t. P_{t-1} as an expectation w.r.t. the pair $(x; w)$:

$$\int \frac{Z_{P_t}}{Z_{P_t \;\; 1}} h(x') P_t(dx') = \mathsf{E}_{x\sim P_t \;\; 1, u_K \sim Q_K}[\hat{w} \;\; h(\;\; 1(f_K(x; u_K)))] = \mathsf{E}_{(x,w)}[\frac{1}{Z_{P_t \;\; 1}} \; w \; \mathsf{E}_{u_K \sim Q_K}[\hat{w} \;\; h(\;\; 1(f_K(x; u_K)))]]:$$

By linearity of expectation, we can rewrite to get:

$$\frac{1}{Z_{P_t \;\; 1}} \int Z_{P_t} h(x') P_t(dx') = \frac{1}{Z_{P_t \;\; 1}} \mathsf{E}[w \;\; \hat{w} \;\; h(\;\; 1(f_K(x; u_K)))]:$$

Canceling the $\frac{1}{Z_{P_t \;\; 1}}$ and rewriting $Z_{P_t} P_t$ as $\tilde{P}_t$, we get the desired equation, that $\mathsf{E}[w \;\; \hat{w} \;\; h(x')] = \int h(x') \tilde{P}_t(dx')$. □

B Proof of SMCP^3 Convergence

B.1 SMCP^3 moves define Feynman Kac models

To reuse standard SMC convergence arguments from the literature, like Proposition 2, we first prove that Algorithm 1 implements SMC for a certain Feynman-Kac model. We first define Feynman-Kac models, following (6), but using slightly different notation and terminology for consistency with this paper.

Definition 5 (Feynman-Kac model.)**.** A Feynman-Kac model is a 4-tuple $(\mathsf{Q}_1; (Q_t)_{t=2}^T; G_1; (G_{t=t}^T))$, where

1. Q_1 is a probability measure on a measurable space X_1
2. $Q_t : X_{t-1} \;\; ! \;\; X_t$ is a kernel between 2 measurable spaces
3. $G_1 : X_1 \;\; ! \;\; \mathsf{R}_{\geq 0}$ is a measurable "potential" function on X_1
4. $G_t : X_{t-1} \quad X_t \;\; ! \;\; \mathsf{R}_{\geq_0}$ is a measurable "potential" function on measurable product space $X_{t-1} \quad X_t$

This model is said to *target* the sequence of measures $(\mathsf{P}_t)_{t=1}^T$ on $(X_t)_{t=1}^T$, where for any measurable $A \quad X_1$,

$$\mathsf{P}_1(A) = \int_A G_1(x_1) \mathsf{Q}_1(dx_1) \tag{1}$$

and for any $t > 1$, for any measurable $A \quad X_t$,

$$\mathsf{P}_t(A) = \int_{\mathcal{X}_t \;\; 1} \int_A G_t(x_{t-1}; x_t) Q_t(x_{t-1} \;\; ! \;\; dx_t) \mathsf{P}_{t-1}(dx_{t-1}): \tag{2}$$

Note that in the definition of a Feynman-Kac model from (6), each space X_t must be the same space X. This restriction is immaterial because given any Feynman-Kac model defined as above, we can obtain a Feynman-Kac model under the definition from (6) by taking $X = \;\; {}_{t=1}^T X_t$, and extending Q_1 and G_1 and each Q_t and G_t to measures/kernels/functions on this full space. Note also that what we call Q_1 and Q_t, (6) calls M_0 and M_{t-1}, and what we call P_t, (6) calls Q_{t-1}.

Theorem 3 (The Feynman-Kac model yielded by SMCP^3)**.** Consider any sequence $(\tilde{P}_t)_{t=1}^T$ of finite measures on measurable spaces $X_1; : : : ; X_T$ admitting densities $\tilde{p}_t$ w.r.t base measures $\;\; _t$ on each X_t. Let Q_1 be a measure on X_1 s.t. $\tilde{P}_1 << Q_1$, admitting density q_1 w.r.t. $\;\; _1$. Let $(K_t; L_t)_{t=2}^T$ be a sequence s.t. $(K_t; L_t)$ is an SMCP^3

move from $\tilde{P}_{t-1}$ to $\tilde{P}_t$ (Def. 2), where $K_t = (U_{K_t}; Q_{K_t}; f_{K_t})$ and $L_t = (U_{L_t}; Q_{L_t}; f_{L_t})$, and where Q_{K_t} and Q_{L_t} admit densities q_{K_t} and q_{L_t} w.r.t. measures $\quad_{K_t}$ and $\quad_{L_t}$.

Then there exists a Feynman-Kac model $(Q_1; (Q_t)_{t=2}^T; G_1; (G_{t=t}^T)$ targetting the sequence $(P_t)_{t=1}^T$, where $P_1 = \tilde{P}_1$ and $\forall t > 1; P_t = \tilde{P}_t \quad Q_{L_t}$. In particular, one such Feynman-Kac model is the one with components as follows:

1. $Q_1 = Q_1$
2. $Q_2(x_1 \;!\; A) = \bar{Q}_2(x_1 \;!\; A)$ and $\forall t > 2; Q_t((x_{t-1}; u_{t-1}) \;!\; A) = \bar{Q}_t(x_{t-1} \;!\; A)$
3. $G_1(x_1) = \tilde{p}_1(x_1) / q_1(x_1)$
4. $G_2(x_1; (x_2; u_{L_2})) = g_2(x_1; (x_2; u_{L_2}))$ and $\forall t > 2; G_t((x_{t-1}; u_{L_{t-1}}); (x_t; u_{L_t})) = g_t(x_{t-1}; (x_t; u_{L_t}))$

In the above, $\bar{Q}_t(x_{t-1} \;!\; \quad)$ is the kernel from $X_{t-1} \;!\; X_t \quad U_{L_t}$ implemented by sampling $u_{K_t} \quad Q_{K_t}(x_{t-1} \;!\; \quad)$ then running f_{K_t},

$$\bar{Q}_t(x_{t-1} \;!\; A) = \int_{U_{K_t}} 1_{f_{K_t}(x_{t-1}, u_{K_t}) \in A} Q_{K_t}(x_{t-1} \;!\; du_{K_t});$$

and $g_t : X_{t-1} \quad U_{L_t} \;!\; \mathbb{R}_{\geq 0}$ is the function yielding the importance-weight update from Theorem 1 for the tth SMCP^3 move,

$$g_t(x; (x'; u_L)) = \frac{\tilde{p}_t(x') q_{L_t}(x' \;!\; u_L)}{\tilde{p}_{t-1}(x) q_{K_t}(x \;!\; u_K)} \quad \frac{d(\quad_t \quad\quad_{L_t})}{d((\quad_{t-1} \quad\quad_{K_t}) \quad f_{K_t}^{-1})}(x'; u_L) \qquad \text{where } (\_; u_K) = f_{L_t}(x'; u_L):$$

Proof. Equation 1 certainly holds since Q_1 and G_1 implement an importance sampler for $\tilde{P}_1$. Thus all we need is to verify that Equation 2 holds. Consider any measurable $A \quad X_t \quad U_{L_t}$. The target measure on this set is

$$P_t(A) = (\tilde{P}_t \quad Q_{L_t})(A) := \int_{X_t} \int_{U_{L_t}} 1_{(x_t, u_{L_t}) \in A} Q_{L_t}(x_t \;!\; du_{L_t}) \tilde{P}_t(dx_t)$$

Writing $\bar{P}_t(A)$ to refer to the R.H.S. of Eq. 2, so our goal is to show $P_t = \bar{P}_t$, we have

$$\begin{aligned}
\bar{P}_t(A) &= \int_{X_{t-1} \times U_{L_{t-1}}} \int_A g_t(x_{t-1}; (x_t; u_{L_t})) \bar{Q}_t(x_{t-1} \;!\; dx_t; du_{L_t}) P_{t-1}(dx_{t-1}; du_{L_{t-1}}) \\
&= \int_{X_{t-1}} \int_{U_{L_{t-1}}} \int_A g_t(x_{t-1}; (x_t; u_{L_t})) \bar{Q}_t(x_{t-1} \;!\; dx_t; du_{L_t}) Q_{L_{t-1}}(x_{t-1} \;!\; du_{L_{t-1}}) \tilde{P}_{t-1}(dx_{t-1}) \\
&= \int_{X_{t-1}} \int_A g_t(x_{t-1}; (x_t; u_{L_t})) \bar{Q}_t(x_{t-1} \;!\; dx_t; du_{L_t}) \tilde{P}_{t-1}(dx_{t-1}) \\
&= \int_{X_{t-1}} \int_{U_{K_t}} 1_{f_{K_t}(x_{t-1}, u_{K_t}) \in A} g_t(x_{t-1}; f_{K_t}(x_{t-1}; u_{K_t})) \tilde{p}_{t-1}(x_{t-1}) Q_{K_t}(x_{t-1} \;!\; du_{K_t}) \quad_{t-1}(dx_{t-1}) \\
&= \int_{X_{t-1}} \int_{U_{K_t}} 1_{f_{K_t}(x_{t-1}, u_{K_t}) \in A} g_t(x_{t-1}; f_{K_t}(x_{t-1}; u_{K_t})) q_{K_t}(x_{t-1} \;!\; u_{K_t}) \tilde{p}_{t-1}(x_{t-1}) \quad_{K_t}(du_{K_t}) \quad_{t-1}(dx_{t-1}) \\
&= \int_{f_{K_t}^{-1}(A)} \frac{d(\quad_t \quad\quad_{L_t})}{d((\quad_{t-1} \quad\quad_{K_t}) \quad f_{K_t}^{-1})}(f_{K_t}(x_{t-1}; u_{K_t}))(\quad_{t-1} \quad\quad_{K_t})(dx_{t-1}; du_{K_t}) \\
&\qquad q_{K_t}(x_{t-1} \;!\; u_{K_t}) \tilde{p}_{t-1}(x_{t-1}) \quad \frac{\tilde{p}_t(x_t) q_{L_t}(x_t \;!\; u_{L_t})}{q_{K_t}(x_{t-1} \;!\; u_{K_t}) \tilde{p}_{t-1}(x_{t-1})} \quad \text{where } (x_t; u_{L_t}) = f_{K_t}(x_{t-1}; u_{K_t}) \\
&= \int_A \frac{d(\quad_t \quad\quad_{L_t})}{d((\quad_{t-1} \quad\quad_{K_t}) \quad f_{K_t}^{-1})}(x_t; u_{L_t})((\quad_{t-1} \quad\quad_{K_t}) \quad f_{K_t}^{-1})(dx_t; du_{L_t}) \quad \tilde{p}_t(x_t) q_{L_t}(x_t \;!\; u_{L_t}) \\
&= \int_A \tilde{p}_t(x_t) q_{L_t}(x_t \;!\; u_{L_t}) d(\quad_t \quad\quad_{L_t})(dx_t; du_{L_t}) \\
&= \int_{X_t} \int_{U_{L_t}} 1_{(x_t, u_{L_t}) \in A} Q_{L_t}(x_t \;!\; du_{L_t}) \tilde{P}_t(dx_t) \\
&= P_t(A)
\end{aligned}$$

□

B.2 Proof of Proposition 2

Proposition 2. For any $t \in \{1,\dots,T\}$, let $(w_n^t; x_n^t)_{n=1}^{\infty}$ be the particle cloud generated by Alg. 1 at timestep t. If $\tilde{P}_t$, K_t, L_t, and Q_1 are such that Alg. 1's incremental weights $\hat{w}_i^t$ are bounded above, then for any continuous, bounded function $\varphi : X_t \to \mathbb{R}$, there exists σ s.t.

$$\sqrt{N}\left(\frac{1}{N}\sum_{n=1}^{N} w_n^t \varphi(x_n^t) \quad \int_{X_t} \varphi(x)\tilde{P}_t(dx)\right) \xrightarrow{D} \mathcal{N}(0; \sigma)$$

as $N \to \infty$, where $\xrightarrow{D}$ is convergence in distribution.

Proof. Let $u_{L_t}^n$ denote the u_L values generated for each particle at time t by Alg. 1. By Theorem 3 above and Proposition 11.2 in (6), for any SMCP^3 algorithm in which the incremental importance weights are upper bounded at each timestep, for any continuous function $\varphi'' : X_t \times U_{L_t} \to \mathbb{R}$, for some $\sigma > 0$,

$$\sqrt{N}\left(\frac{1}{N}\sum_{n=1}^{N} w_n^t \varphi''(x_n^t; u_{L_t}^n) \quad \int_{X_t} \varphi''(x; u_{L_t})(\tilde{P}_t \otimes Q_{L_t})(dx; du)\right) \xrightarrow{D} \mathcal{N}(0; \sigma)$$

All that remains is to observe that any continuous bounded $\varphi : X_t \to \mathbb{R}$ can be extended to a continuous, bounded function $\varphi'' : X_t \times U_{L_t} \to \mathbb{R}$ (by $\varphi''(x; u) = \varphi(x)$), and that

$$\int \varphi''(x; u)(\tilde{P}_t \otimes Q_{L_t})(dx; du) = \int \varphi(x)\tilde{P}_t(dx).$$

□

C The Measurable Space of Traces

For ease of presentation, we assume Gen has just a handful of basic value types over which primitive distributions are defined: the reals $\mathbb{R}$, the natural numbers $\mathbb{N}$, and the Booleans $\mathbb{B}$. For each such type τ, we choose a measure space $(V_\tau; \mathcal{V}_\tau; \nu_\tau)$ of values, where ν_τ is a reference measure. For the reals, we choose $\nu_{\mathbb{R}} = \Lambda$ (the Lebesgue measure), and for discrete types we choose the counting measure. We let K denote a countable set of *names* for random variables (e.g., the strings). Every execution of a program encounters some set of sampling statements, each with a name and a distribution over some type, and we call these paths *trace shapes*:

Definition 6. A *trace shape* $s \in \mathcal{S}$ is a finite set of entries $(k; \tau)$, such that $k \in K$ is a name (and no two entries share the same name), and τ is a type.

A trace is a trace shape, together with its values:

Definition 7. A *trace* $t \in \mathcal{T}$ is a trace shape s and a tuple $\mathbf{v} \in \bigotimes_{(k,\tau)\in s} V_\tau$, with one value for each name in s.

We equip $\mathcal{T}$ with the disjoint union σ-algebra, where the union is taken over the countable set of trace shapes, and each element of the union is a product space $\bigotimes_{(k,\tau)\in s} V_\tau$. For each trace shape s, we can define the product reference measure $\nu_s = \bigotimes_{(k,\tau)\in s} \nu_\tau$. Then define reference measure $\nu_{\mathcal{T}}$ over all traces as

$$\nu_{\mathcal{T}}(B) = \sum_{s\in\mathcal{S}} \nu_s(\{\mathbf{v} \mid (s; \mathbf{v}) \in B\}).$$

D Proof of Theorem 2

D.1 PAP functions

The statement of the Theorem includes the assumption that a certain function f is PAP, or piecewise-analytic-under-analytic-partition. We first review Lee et al. (29)'s definition of PAP functions on Euclidean spaces, then use the tools from Lew et al. (31) to extend the definition to spaces of traces.

Definition 8. Let $U \subseteq \mathbb{R}^n$ and $V \subseteq \mathbb{R}^m$. A function $f : U \to V$ is *(real-)analytic at a point $x \in U$* if it is smooth at x and there exists an open neighborhood I of x on which f is equal to its Taylor series. If f is analytic at every point in its domain U, then we say f is a *(real-)analytic function*.

Definition 9. An *analytic set* $A \subseteq \mathbb{R}^n$ is a set of the form $\{x \in U \mid f_1(x) = 0 \wedge \cdots \wedge f_k(x) = 0\}$, where U is an open subset of $\mathbb{R}^n$, $k \geq 0$ is a finite natural number, and each $f_i : U \to \mathbb{R}$ is analytic on U.

Definition 10. Let $U \subseteq \mathbb{R}^n$ and $V \subseteq \mathbb{R}^m$. A partial function $f : U \rightharpoonup V$ is *piecewise analytic under analytic partition*, or *PAP*, if there exists a countable family $\{(A_i; U_i; f_i)\}_{i \in I}$, where:

- each U_i is an open subset of U,
- each A_i is an analytic subset of U_i,
- the A_i are pairwise disjoint and form a *partition* of the domain of f,
- each $f_i : U_i \to \mathbb{R}^m$ is analytic, and
- for each i, for all $x \in A_i$, $f(x) = f_i(x)$.

Remark 2. If we view the Booleans $\mathbb{B}$ as a subset $\{0, 1\}$ of $\mathbb{R}$, and similarly view the naturals $\mathbb{N}$ and integers $\mathbb{Z}$ as subsets of $\mathbb{R}$, this definition can be applied directly to functions that accept and return vectors holding both continuous and discrete values. (These functions are not defined when their discrete inputs are set to "invalid" values, but Definition 10 applies to partial functions, so this is not an issue.) Using Definition 10, we can characterize which of these functions on hybrid discrete-continuous spaces are PAP. Let D_I be the discrete indices of the input vector $\mathbf{u}$ and D_O be the discrete indices of the output vector $\mathbf{v}$. For any assignment $\mathbf{v}_D$ to the discrete indices of the output vector, let $U_{\mathbf{v}_D} \subseteq U$ to be the preimage $f^{-1}(\{\mathbf{v} \in V \mid \mathbf{v}[D_O] = \mathbf{v}_D\})$. Then, for each assignment $\mathbf{u}_D$ to the discrete components of the input vector $\mathbf{u}$, we can consider the restriction $f^{\mathbf{u}_D}_{\mathbf{v}_D}$ of f to $\{u \in U_{\mathbf{v}_D} \mid u[D_I] = \mathbf{u}_D\}$. Because the discrete inputs and outputs are fixed for every vector in its domain, $f^{\mathbf{u}_D}_{\mathbf{v}_D}$ can really be viewed as a function only of the *continuous* components in the input vector, into the continuous part of the output vector. Unfolding the definition of PAP, it can be shown that f is PAP if and only if, for every $\mathbf{u}_D$ and $\mathbf{v}_D$, $f^{\mathbf{u}_D}_{\mathbf{v}_D}$ is PAP.

The observations in Remark 2 can be generalized to support arbitrary countable unions of Euclidean spaces:

Definition 11. Let J, K be countable sets and let $X_j \subseteq \mathbb{R}^{n_j}$ for each $j \in J \cup K$. A partial function $f : \coprod_{j \in J} X_j \rightharpoonup \coprod_{k \in K} X_k$ is PAP if, for each $j \in J$ and $k \in K$, the function $f_{j,k} : \{x \in X_j \mid \exists y \in X_k. f((j, x)) = (k, y)\} \to X_k$ mapping x to $\pi_2(f((j, x)))$ is PAP.

Recall that $\mathbb{T} = \coprod_{s \in \mathbb{S}}(\prod_{(k,\tau) \in s} V_\tau)$. Since each $V_\tau \subseteq \mathbb{R}$, we can view $\mathbb{T}$ as a particular subset of $\coprod_{s \in \mathbb{S}} \mathbb{R}^{|s|}$, and similarly $\mathbb{T} \times \mathbb{T}$ as a subset of $\coprod_{(s_1,s_2) \in \mathbb{S} \times \mathbb{S}} \mathbb{R}^{|s_1|+|s_2|}$. Supposing U and V are subsets of $\mathbb{T} \times \mathbb{T}$, a function $f : U \to V$ can be viewed as a partial function $\hat{f} : \coprod_{(s_1,s_2) \in \mathbb{S} \times \mathbb{S}} \mathbb{R}^{|s_1|+|s_2|} \rightharpoonup \coprod_{(s_1,s_2) \in \mathbb{S} \times \mathbb{S}} \mathbb{R}^{|s_1|+|s_2|}$, and we can apply our definitions above to establish whether it is PAP.

Remark 3. We now work out the implications of this definition for functions f that accept and return pairs of traces. The "discrete data" $\mathbf{t}_D$ of a pair of traces $\mathbf{t} = (t_1, t_2)$ is a pair of trace shapes (s_1, s_2) and a pair of assignments (v^1_D, v^2_D) to just the discrete parts of each trace. Fixing $\mathbf{t}_D$, the continuous data of the pair is just a vector of reals, concatenating the real values from the first trace to the real values from the second trace. Write $n_C(\mathbf{t}_D)$ for the total number of continuous values in the trace pair. For each possible input discrete data $\mathbf{t}^I_D$ and output discrete data $\mathbf{t}^O_D$, there is a partial function $f^{\mathbf{t}^I_D}_{\mathbf{t}^O_D} : \mathbb{R}^{n_C(\mathbf{t}^I_D)} \rightharpoonup \mathbb{R}^{n_C(\mathbf{t}^O_D)}$ that accepts as input the continuous data for an input trace pair (whose discrete data is $\mathbf{t}^I_D$), and if $f(\mathbf{t}^I)$ matches the output discrete data $\mathbf{t}^O_D$, outputs the continuous data of $f(\mathbf{t}^I)$ (otherwise, it is undefined). Then f is PAP if each of these partial functions $f^{\mathbf{t}^I_D}_{\mathbf{t}^O_D}$ is PAP.

Remark 4. Suppose U and V are subsets of $\mathbb{T} \times \mathbb{T}$ and $f : U \to V$ is a PAP *bijection*, with PAP inverse $f^{-1} : V \to U$. Then each $f^{\mathbf{t}^I_D}_{\mathbf{t}^O_D}$ is PAP and is the inverse of $(f^{-1})^{\mathbf{t}^O_D}_{\mathbf{t}^I_D}$.

D.2 Change-of-variables for PAP bijections on subsets of T T.

Lemma 2. Let ; be measures over X and Y, and $f : X \to Y$ a measurable bijection, such that (f^{-1}). Let $fA_i g_{i\in I}$ be a countable family of subsets of X such that $X \setminus [_{i\in I} A_i$ has -measure-zero. Further, let $g : Y \to \mathsf{R}_{\geq 0}$ be such that for each i, for all $y \in f(A_i)$, $g(y) = \frac{d\nu_i}{d(\mu_I \circ f_i^{-1})}(y)$, where $_i$ is the restriction of to A_i, $_i$ is the restriction of to $f(A_i)$, and $f_i : A_i \to f(A_i)$ is the restriction of f to A_i. Then g is a Radon-Nikodym derivative of with respect to f^{-1}.

Theorem 2. Let T_1^2 and T_2^2 be measurable subsets of T T, and let $_1$ and $_2$ be restrictions of T T to these subsets. Suppose $f : \mathsf{T}_1^2 \to \mathsf{T}_2^2$ is a PAP bijection and that $_2$ is absolutely continuous with respect to $_1$ f^{-1}. Then

$$\frac{d\ _2}{d(\ _1\ f^{-1})}(f(\ _1;\ _2)) = |\det(J f_{\tau_1,\tau_2}(\ (\ _1) +\!\!+\ (\ _2))|;$$

where : T $\to$ $t_{d\in\mathsf{N}}\mathsf{R}^d$ extracts all the real-valued entries in a trace into a vector, $_\tau : \mathsf{R}^{|\rho(\tau)|} \to$ T replaces the real entries in with values from a vector, and $f_{\tau_1,\tau_2} : \mathsf{R}^{|\rho(\tau_1)|+|\rho(\tau_2)|} \to \mathsf{R}^{|\rho(\tau_1)|+|\rho(\tau_2)|}$ accepts as input the concatenation of $\mathbf{v}_1 \in \mathsf{R}^{|\rho(\tau_1)|}$ and $\mathbf{v}_2 \in \mathsf{R}^{|\rho(\tau_2)|}$, computes $'_1; '_2 = f(\ _{\tau_1}(\mathbf{v}_1);\ _{\tau_2}(\mathbf{v}_2))$, and then returns the vector concatenation $(\ '_1) +\!\!+\ (\ '_2)$.

Proof. We will use Lemma 2 to carve T_1^2 into pieces, and prove the result separately for each piece.

For each $\mathbf{t}_D^I; \mathbf{t}_D^O$, we have from Remark 4 that $f^{\mathbf{t}_D^I}_{\mathbf{t}_D^O}$ is a PAP bijection from $U \subseteq \mathsf{R}^n \to V \subseteq \mathsf{R}^m$, for some n and some m. Then there is some partition of U into analytic subsets A_j of R^n; let $J_{(\mathbf{t}_D^I,\mathbf{t}_D^O)}$ be the index set of this partition. Then let $I = \{(\mathbf{t}_D^I; \mathbf{t}_D^O; j) \mid j \in J_{(\mathbf{t}_D^I,\mathbf{t}_D^O)} \wedge \Lambda(A_j^{(\mathbf{t}_D^O,\mathbf{t}_D^I)}) > 0\}$, and define $\mathsf{T}^2_{(\mathbf{t}_D^I,\mathbf{t}_D^O,j)}$ the set of pairs of traces with discrete data $\mathbf{t}_D^I$ and continuous data in $A_j^{(\mathbf{t}_D^O,\mathbf{t}_D^I)}$. By Lemma 2, it suffices to show that for each $(\mathbf{t}_D^I; \mathbf{t}_D^O; j)$,

$$\frac{d\ |_2 j_{f(\mathsf{T}^2_{(\mathbf{t}_D^I;\mathbf{t}_D^O;j)})}}{d(\ _1 j_{\mathsf{T}^2_{(\mathbf{t}_D^I;\mathbf{t}_D^O;j)}}\ f^{-1}_{(\mathbf{t}_D^I,\mathbf{t}_D^O,j)})}(f(t_1; t_2)) = |\det(J f_{t_1,t_2}(\ (t_1) +\!\!+\ (t_2)))|;$$

First, note that restricted to $\mathsf{T}^2_{(\mathbf{t}_D^I,\mathbf{t}_D^O,j)}$, $g(t_1; t_2) := (t_1) +\!\!+ (t_2)$ is a bijection: the discrete data of $(t_1; t_2)$ is fixed to $\mathbf{t}_D^I$, and so even though g deletes this information by extracting only the continuous values from the traces, it is injective and can be inverted by reattaching the discrete data $\mathbf{t}_D^I$. Indeed, for any $(t_1; t_2)$ with discrete data $\mathbf{t}_D^I$, $f_{t_1,t_2} = g \circ f \circ g^{-1}$. So we have:

$$\begin{aligned}
\frac{d\ |_2 j_{f(\mathsf{T}^2_{(\mathbf{t}_D^I;\mathbf{t}_D^O;j)})}}{d(\ _1 j_{\mathsf{T}^2_{(\mathbf{t}_D^I;\mathbf{t}_D^O;j)}}\ f^{-1}_{(\mathbf{t}_D^I,\mathbf{t}_D^O,j)})}(f(t_1; t_2)) &= \frac{d(\ |_2 j_{f(\mathsf{T}^2_{(\mathbf{t}_D^I;\mathbf{t}_D^O;j)})}\ g^{-1})}{d(\ _1 j_{\mathsf{T}^2_{(\mathbf{t}_D^I;\mathbf{t}_D^O;j)}}\ f^{-1}_{(\mathbf{t}_D^I,\mathbf{t}_D^O,j)}\ g^{-1})}(g(f(t_1; t_2))) \\
&= \frac{d(\ |_2 j_{f(\mathsf{T}^2_{(\mathbf{t}_D^I;\mathbf{t}_D^O;j)})}\ g^{-1})}{d((\Lambda\ (g^{-1})^{-1})\ f^{-1}_{(\mathbf{t}_D^I,\mathbf{t}_D^O,j)}\ g^{-1})}(g(f(t_1; t_2))) \\
&= \frac{d\Lambda}{d(\Lambda\ (f_{t_1,t_2} j_{g(\mathsf{T}^2_{(\mathbf{t}_D^I;\mathbf{t}_D^O;j)})})^{-1})}(f_{t_1,t_2}(g(t_1; t_2)));
\end{aligned}$$

where the first line uses the fact that g is a bijection, the second and third use the fact that $_1$ $(g^{-1})^{-1}$ is the Lebesgue measure on the image (under g) of $_1$'s support,[9] and the third also uses the fact that $f_{t_1,t_2} = g \circ f \circ g^{-1}$. Since f is PAP, $f_{t_1,t_2} j_{g(\mathsf{T}^2_{(\mathbf{t}_D^I;\mathbf{t}_D^O;j)})}$ is real-analytic, and we can apply the standard change-of-variables formula for pushforwards of the Lebesgue measure by differentiable bijections, yielding the desired Jacobian determinant.

□

[9] This is true because we defined $_{\mathsf{R}} := \Lambda$ when defining the reference measure $_{\mathsf{T}}$ for traces.

E Standard PPL automation

Sampling and evaluating densities of traces. The Gen probabilistic programming system (10) automatically generates procedures for *sampling* traces, *evaluating densities* of traces, and *computing return values* from traces. The density computed by Gen is precisely the density of a program's sampling distribution over traces with respect to the reference measure $\quad_T$. Roughly, these procedures work by overriding the behavior of *fnameg* `distribution` statements:

- To `SAMPLE-TRACE`, initialize storage for an empty trace, and run the program. At each *fnameg* `distribution` statement, draw a sample from the distribution and record it (and its type) in a trace under `name`. When the program is finished, return both the trace and the return value.
- To `EVAL-LOGPDF` of a given trace t, initialize a weight to 1, and at each *fnameg* `distribution` statement, lookup the value v associated with the key `name` in the trace t, and multiply the weight by the density of `distribution` at v. If any `name` is missing, or if after finishing execution the trace contains unvisited `name`s, return 0; otherwise, return the final weight.
- To `COMPUTE-RETVAL` given a trace t, run the program but instead of sampling, use the values already in the trace (as for density evaluation). Once finished executing, return what the program returns.

F Experimental Details

All experiments were run on a commodity laptop with 32GB of RAM, using 8 cores. Every experiment presented in Sec. 4 ran in under 10 minutes on our hardware.

F.1 Online inference in state-space models

F.1.1 Inference in a state-space model with a differentiable likelihood

This model, and the baseline inference algorithms we compare against, are described in section 4.1

SMCP^3 inference algorithm. Our SMCP^3 inference algorithm for this model is illustrated Fig. 4, and defined in Gen pseudocode in Fig. 2. The space U_{K_t} of auxiliary randomness is R and the space of U_{L_t} is $f;g$ (there is no auxiliary randomness for the L proposal). Algorithm 3 gives the K and L kernels for this algorithm in mathematical notation. Given particle $x_{1:t-1}$, the K kernel first samples an auxiliary u from the dynamics prior, then samples x_t via a step of Langevin ascent starting from u. It returns the updated particle $x_{1:t} = (x_{1:t-1}; x_t)$, and the only element of the auxiliary space U_{L_t}. The L kernel simply samples $u \quad P(\ ; x_{t-1})$ (ignoring x_t).

Algorithm 3 SMCP^3 kernels for Langevin particle extension.

Require: Langevin ascent step-size $\quad$.
Require: Observation trajectory $y_{1:t}$.
procedure K($x_{1:t-1}$)
  $u \quad P(X_t = \ ; x_{t-1})$. Sample an auxiliary position from the dynamics prior
  . Sample x_t via a step of Langevin ascent, starting from u.
  $x_t \quad N(u + {}^{2}\frac{\partial \log P(X_t = u, y_t | x_{t\ 1})}{\partial u}); \ p\overline{2}\ )$
  return $((x_{1:t-1}; x_t); \ ;)$
end procedure
procedure L($x_{1:t}$)
  $u \quad P(X_t = \ j x_{t-1})$
  return $(x_{1:t-1}; u)$
end procedure

Experiment details. Fig. 3 shows an artificial observation trajectory generated by adding $N(0; 0{:}8)$ noise to each position in the hand-written vector $[[0; 3; 6; 9; 12; 9; 12; 16; 18; 18; 20; 17]$. (Fig 5b shows model-average

results.). For this trajectory, fig. 3 shows the 1- band of the exact filtering posterior distribution over latent states at each timestep (obtained via Kalman filtering), and 100 resampled particles and log-marginal-likelihood estimates from our SMCP^3 inference algorithm, resample-move SMC using MALA rejuvenation, and the boostrap particle filter, each run with 4 particles. For the SMCP^3 algorithm and MALA, we use Langevin-ascent step size = 0:3.

Fig. 5a shows 200 particles generated at $t = 1$ by each inference algorithm, given $y_1 = 5$. Each particle is rendered at its x_1 position, and at a y coordinate in the figure so it lands on the posterior curve for $P(X_1 | y_1)$. Fig 5b shows log-marginal-likelihood estimates from each inference algorithm, averaged at each particle count over 200 runs each of 20 random 10-step observation sequences generated from the model. All experiments use multinomial resampling triggerred when the effective sample size falls below 1=5 the number of particles.

F.1.2 Inference in a state-space model with a rendering-based likelihood

Figure 6 shows inference results in a simple state-space model for tracking 3D objects from depth data, comparing an SMCP^3 algorithm (which is a variant of the algorithm from the previous section, but replacing the ULA step with a grid-enumeration step because the likelihood is non-differentiable) to the bootstrap particle filter.

Model. Our motion model is used for tracking a single degree of freedom of the position of a cube in 3D space. The position of the cube is $(x_t; 0; 0)$ where x_t is a latent state that evolves according to Gaussian random walk dynamics, $x_1 = 0$; $8t > 1; x_t \quad N(x_{t-1}; 0:7)$. The observed data y_t is a "point-cloud" of 200 3D positions detected by a depth-camera (so $y_t \; 2 \; \mathbb{R}^{3\times 200}$). The likelihood $P(y_t | x_t)$ is defined using the following generative process. First, a deterministic renderer traces M rays from the camera to the scene, and records the coordinates at which these rays first intersect the cube, producing a point cloud $q_t = [q_t^1; : : : ; q_t^M] \; 2 \; \mathbb{R}^{3\times M}$. Then, the observed point-cloud y_t is generated from the following Gaussian mixture model:

$$p(y_t | q_t^1; : : : ; q_t^M) = \prod_{j=1}^{200} \sum_{i=1}^{M} \frac{1}{M} N(y_t^j; q_t^i; 10^{-2} I)$$

where y_t^j denotes the j-th column of y_t.

SMCP^3 inference algorithm. Alg. 4 gives the K and L proposals used by our SMCP^3 algorithm. The K kernel first proposes a position u from the dynamics model, then chooses one of a finite number of points on a grid centered at u as the value for x_t; the L kernel fills in the value of u by enumerating each one which could have led to K returning x_t.

Experiment details. Figure 6 shows frames of a real RGB-D video in which a box is sliding across the floor, and inferred latent states from a simple enumeration-based inference algorithm, to illustrate the setup using real data. The plot in figure 6 is generated using a synthetic observation trajectory generated from the model prior. The cube side length was set to 0:5 units and the camera was positioned at $(4; 0; 1:4)$ and oriented toward the origin. The plot shows log marginal data likelihood results averaged across 10 independent inference runs from the baseline and the SMCP3 inference algorithm.

F.2 Online inference in mixture models

We first give further details on our DPMM model, the baseline inference algorithms, and the details of the experiments showcased in Sec. 4.2. We then detail the SMCP^3 inference algorithm.

F.2.1 Model, Baselines, and Experiment Details

Section 4.2 describes the DPMM we use for online clustering. It has two parameters: (1) the parameter $\quad_D$ of the Dirichlet process, and (2) the exchangeable likelihood F, which determines the distribution over data produced by a single cluster. Specifically, for any F, there is a measurable value space $(V; V; \quad_V)$, and $F : \quad_{n\in\mathbb{N}} V^n \; ! \; \mathbb{R}_{\geq 0}$ is the function s.t. $F(\vec{v})$ is the joint probability density of vector $\vec{v}$ w.r.t. $\quad_V^{|\vec{v}|}$. (F restricted to the domain V^n is a probability density over n-long vectors; F is not a probability density over the whole space $\quad_{n\in\mathbb{N}} V^n$.)

We run inference in an online manner, meaning that data is streamed into the inference algorithm over time, and after seeing each subset $y_{1:t}$ of the data, the inference algorithm must output an approximation of $P(\Pi_t | y_{1:t})$,

Algorithm 4 SMCP^3 kernels for enumeration-based particle extension.

Require: Grid step size and range N.
Require: Observation trajectory $y_{1:t}$.

procedure K($x_{1:t-1}$)
  u $P(\ ; x_{t-1})$. Sample an auxiliary position from the dynamics prior
  . Enumerate over the grid, starting from u, and evaluate the probability of each state.
  D fg . Initialize score dictionary.
  for $i =$ $N; : : : ; N$ **do**
    x_t^i $u + i$. Potential x_t value in grid.
    s $P(x_t^i | x_{t-1}) P(y_t | x_t^i)$. Joint probability score for this x_t value.
    D D [$f x_t^i$ 7! $s g$
  end for
  Sample x_t P_D, where P_D is the distribution so $P_D(x_t)$ / $D[x_t]$.
  return $((x_{1:t-1} : : : ; x_t); \ ;)$
end procedure
procedure L($x_{1:t}$)
  D fg . Initialize score dictionary.
  for $i =$ $N; : : : ; N$ **do**
    u^i $x_t + i$. Potential u value in grid.
    s $P(u | x_{t-1})$
    D D [$f x_t^i$ 7! $s g$
  end for
  Sample u P_D, where P_D is the distribution so $P_D(u)$ / $D[u]$.
  return $(x_{1:t-1}; u)$
end procedure

the posterior over possible partitions of data-indices $f1; : : : ; tg$, given the first t datapoints.

Mixture-model likelihoods. We use three different F likelihoods in our experiments, two for modeling real data (so $V = \mathsf{R}$, with the regular sigma-algebra and Lebesgue measure) and one for modeling string data (so V is the set of strings, equipped with the discrete sigma algebra and counting measure).

The first data-likelihood, F_1, models a Gaussian data-cluster with an unknown mean $N(\ _0; \ ^2_0)$ and a known variance ^{2}. For any vector y,

$$F_1(y) = \int_{\mathsf{R}} \prod_{y \in \vec{y}} N(y; \ ; \ ^2) N(d \ ; \ _0; \ ^2_0):$$

The second data-likelihood, F_2, models a Gaussian data-cluster with an unknown mean and an unknown variance ^{2}. Parameters $_y; \ ; \ ;$ and are introduced to control the prior over the cluster mean and variance. For any vector y,

$$F_2(y) = \int_{\mathsf{R}} \int_{\mathsf{R}} \sum_{y \in \vec{y}} N(y; \ ; \ ^2) \Gamma(d(\ ^{-2}); \ _y; \ ) N(d \ ; \ ; \ ^{-2}):$$

The third data-likelihood, F_3, models a cluster of strings, by assuming there is some "ground-truth" string s^*, and every string in the cluster is generated by applying a randomly-chosen number of "typos" to s^*. (As with the and parameters above, the string s^* is marginalized out in the likelihood function.) We use the exact likelihood defined in (32) Sec. 5.2, we refer the reader there for details.

Locally-optimal single-datapoint SMC inference. The first inference baseline we experiment is roughly analogous to an optimal particle filter in a state-estimation application. This SMC algorithm uses the following proposal distribution to update a particle Π_{t-1} (a partition of $f1; : : : ; t \ \ 1g$) to a partition Π_t of $f1; : : : ; tg$, given datapoint y_t. The proposal distribution either creates a new cluster containing only t, outputting $\Pi_t = \Pi_{t-1}$ [$f f t g g$, or adds t to an existing $C_* \ 2 \ \Pi_{t-1}$ outputting $\Pi_t = \Pi_{t-1} \ n \ f C_* g$ [$f C_*$ [$f t g g$. This proposal Q chooses among these $j\Pi_{t-1}j + 1$ options such that $Q(\Pi)$ / $p_t(\Pi; y_{1:t})$, where p_t is the PDF of the probabilistic program defining the DPMM, given argument t, at partition Π and data vector $y_{1:t}$.

Resample-move inference algorithm. As a second baseline, we turn the locally-optimal single-datapoint SMC algorithm into a resample-move algorithm using an MCMC kernel. The MCMC kernel is a split/merge kernel, that randomly selects two "chaperone" datapoints (42), and proposes a split (if they are in the same cluster) or a merge (if they are in different clusters). To propose a split, a Gibbs scan on cluster assignments is performed, processing points in sorted order to incorporate them into one of the two new clusters, seeded with the chaperones (this Gibbs scan is done identically to in Q_{split} in the SMCP3 algorithm described below). A Metropolis-Hastings correction is applied to ensure the kernel is stationary.

SMCP^3 inference algorithm. Our SMCP^3 algorithm is fairly sophisticated, so we devote a whole subsection (Sec. F.2.2) to it below.

Experiment details. Table 1 shows the results of mixture-model inference in two real-world datasets: a standard benchmark dataset of real-valued Galaxy velocities (8, 18), and a data-cleaning dataset of 1k strings from Medicare records (30).

To model the Galaxy dataset, we used likelihood F_2 with $_y = 0{:}$, $= 1{=}100$, $= 1{=}2$, $= 1{=}2$, and set the Dirichlet process parameter $_D$ to $1{:}0$. We ran experiments on this dataset using 100-particle SMC; Table 1 shows the mean and empirical standard deviation of the log-marginal-likelihood estimates produced by each of 100 runs of 100-particle SMC, for each inference algorithm. We run inference both with data sorted high-to-low, and with data in a randomly sampled order.

To model the Medicare dataset, we used likelihood F_3, and set $_D = 1{:}0$. Table 1 shows the results of each SMC algorithm on this dataset (using 32 particles for each baseline algorithm, and 2 particles for the SMCP3 algorithm), with means and empirical variances taken over 3 SMC runs for each inference algorithm.

Figure 8 shows results from 10 10-particle SMC inference runs, using the locally-optimal single-datapoint SMC algorithm, and our SMCP^3 algorithm, on synthetically generated data, using likelihood F_1. We generated the dataset shown in the figure from the DPMM with $_D = 1{:}0$, using F_1 with $_0 = 0$, $_0 = 6$, and $= 1$; we ran inference using $_D = 1{:}0$, $_0 = 0$, $_0 = 10^6$, and $= 1$. Because we ran inference using a prior which expects clusters to be much farther away from each other than they are in the presented dataset, on small subsets of the data (e.g. $y_{1:10}$, rather than the full dataset $y_{1:100}$), it is more likely under the posterior that nearby clusters are explained as being a single data-cluster with some outliers. However, as more data is observed, it becomes unlikely to have so many outliers, and hence becomes more likely that there are two surprisingly-nearby clusters. Because the locally-optimal single-datapoint SMC algorithm cannot change which datapoints are clustered together, it produces sub-optimal results on this data. This toy example is intended to illustrate this failure mode —the locally optimal SMC algorithm labeling datapoints which should belong to a new cluster as outliers from an existing cluster —which we also observe occurring in real datasets like the Medicare data. Because our SMCP^3 algorithm is able to split existing clusters into two, it does not suffer from this failure mode.

All experiments are run using multinomial sampling, triggered whenever the effective sample size falls below $1{=}5$ the number of particles.

F.2.2 The SMCP^3 inference algorithm

Our SMCP^3 algorithm extends the above locally-optimal single-datapoint SMC move. Its K proposal first makes this move, producing partition Π_t^1; in the case where this move outputs a partition assigning datapoint t to a new singleton cluster, the K proposal terminates and outputs $\Pi_t = \Pi_t^1$. In the case where Π_t^1 was produced by adding t to an existing cluster C_*, producing $C'_* = C_* \setminus \{t\}$, the K proposal then chooses between 3 ways of producing a final state Π_t from Π_t^1: (1) K may split the cluster C'_* into two new clusters, (2) K may merge C'_* into another cluster $C \in \Pi_t^1 \cap \{C'_*\}$, or (3) K may make no change, and output Π_t^1. Our K only consider split moves in which the index t ends up in a cluster with at least one other datapoint; if C'_* only has 2 elements, split moves are impossible. K makes this decision between splitting, merging, or staying, in the following manner.

Deciding whether to split, merge, or make no change. First, K enumerates every cluster $C \in \Pi_t^1 \cap \{C'_*\}$, and for each of these, computes $s_C = p_t(\Pi_t^C; y_{1:t})$, where $\Pi_t^C = \Pi_t^1 \cap \{C'_*, C\} \cup \{C'_* \cup C\}$ is the partition resulting from merging C'_* with C. These scores give the joint PDF for the partitions resulting from each possible merge move. Second, K computes $s_* = p_t(\Pi_t^1; y_{1:t})$, the joint PDF for the partition resulting from making no change

to the existing partition. Finally K obtains an estimate $\hat{s}_s$ of the total

$$s_s = \sum_{C^0, C^{00}: C^0 \cup C^{00} = C'_*} p_t(\Pi_t^{C^0, C^{00}}; y_{1:t}) \qquad \text{where } \Pi_t^{C^0, C^{00}} = \Pi_t^1 \setminus \{C'_*\} \cup \{C', C''\}.$$

This is the sum over every possible way of splitting C'_* into two clusters C' and C'' of the joint PDF $p_t(\Pi_t^{C^0, C^{00}}; y_{1:t})$ of the partition $\Pi_t^{C^0, C^{00}}$ resulting from making this split. We describe our method of estimating $\hat{s}_s$ below. The final decision of whether to split C'_*, merge C'_* with an existing cluster, or make no change to Π_t^1, is made such that the probability of doing some split move is proportional to $\hat{s}_s$, the probability of making no change is proportional to s_*, and the probability of merging with any particular cluster C is proportional to s_C. (That is, this decision is made by sampling from a categorical distribution over $|\Pi_t^1 \setminus \{C'_*\}| + 2$ possibilities.)

Selecting a particular split, and estimating the total score of all split moves. In the case where K decides to split C'_* into two new clusters, the K kernel must also decide on a particular partition of C'_* into two clusters C' and C''. Our K kernel actually makes this decision *before* deciding whether to split, merge, or stay, at the same time as it produces the estimate $\hat{s}_s$ of the total score of all possible split moves. (Thus, in the case that K does not actually implement a split move, the choice of C' and C'' is still made, and this choice is an auxiliary variable which the L kernel must constrain the value of.) Estimating $\hat{s}_s$ and choosing C', C'' are done via importance sampling using the proposal $Q^K_{\text{split}}(C'_*; y_{1:t} \;!\; u; C', C'')$ described in the following paragraph, which generates a partition C', C'' of C'_*, and also some auxiliary random decisions u. To do importance sampling using a proposal with auxiliary randomness we also introduce a "meta-proposal" kernel $h^K(C'_*; C', C''; y_{1:t} \;!\; u)$ to enable us to "pseudo-marginalize" over the randomness u (our choice of h^K is described below). For a detailed explanation of how meta-proposals can be used to pseudo-marginalize over auxiliary random choices, see (32); note however that no additional theory beyond SMCP^3 is needed to justify this SMCP^3 algorithm or understand the steps it performs. The particular way our K proposal uses Q^K_{split} and h^K is as follows. For some N, for each $i = 1, \ldots, N$, K generates $(C'_i, C''_i, u_i) \quad Q^K_{\text{split}}(C'_*; y_{1:t} \;!\;)$, and computes the importance weight

$$w_i^{\text{split}} = \frac{p_t(\Pi_t^{C^0_i, C^{00}_i}; y_{1:t}) h^K(C'_*; C'_i, C''_i; y_{1:t} \;!\; u)}{Q^K_{\text{split}}(C'_*; y_{1:t} \;!\; C'_i, C''_i; u)}.$$

K then sets $\hat{s}_s = \frac{1}{N} \sum_{i=1}^N w_i^{\text{split}}$. To choose the final proposed C', C'', K samples an index $i \in \{1, \ldots, N\}$ s.t. the probability of choosing any given i is proportional to w_i^{split}, and sets $C', C'' = C'_i, C''_i$. In our experiments, we use $N = 10$.

A smart proposal for a single split move. The kernel $Q^K_{\text{split}}(C'_*; y_{1:t} \;!\; u; C', C'')$ splits C'_* into two clusters using 3 pieces of auxiliary randomness: c_1, c_2, and p. c_1 and c_2 are two "chaperone" datapoints (42) which initialize clusters. p is an additional point that goes in the cluster with c_1. Q^K_{split} deterministically sets $c_1 = t$, then samples a distinct $p \in C'_*$ with probability proportional to $p_2(\{1, 2\}; (y_t, y_p))$, and then samples c_2 uniformly from the remaining points in C'_* not equal to c_1 or p. It initializes two clusters $C' = \{c_1, p\}$ and $C'' = \{c_2\}$. Q^K_{split} then iterates over every remaining point i in C'_* in sorted order (low to high), and for each one, decides to add it either to C' or C'', s.t. the probability of adding it to cluster C' is proportional to the joint PDF of the clustering $C' \cup \{i\}, C''$ with all the datapoints for indices in these sets, and C'' is proportional to the corresponding joint PDF of the clustering $C', C'' \cup \{i\}$. The L kernel (described below) uses a variant of this proposal, Q^L_{split} which also performs this sequential process to assign each point to either C' or C''. Q^L_{split} initializes C' and C'' differently from Q^K_{split}: it first samples c_1 uniformly from C'_*, then samples c_2 uniformly from the remaining points, then sets $p = \quad 1$ deterministically, and finally initializes $C' = \{c_1\}$ and $C'' = \{c_2\}$.

Inverting the auxiliary randomness in a single split proposal. The kernel $h^K(C'_*; C', C''; y_{1:t} \;!\; u)$ must propose values for $u = (c_1, c_2, p)$. It does this by setting $c_1 = t$, sampling p uniformly from $C' \setminus \{t\}$, and sampling c_2 uniformly from C''. The L kernel uses a variant of this, h^L, which sets $p = \quad 1$, samples c_1 uniformly from C', and samples c_2 uniformly from C''.

The L kernel: producing Π_{t-1} from Π_t. The L kernel in our SMCP^3 algorithm must output a partition Π_{t-1}, given the Π_t output by K. If $\{t\} \in \Pi_t$, L simply outputs $\Pi_{t-1} = \Pi_t \setminus \{\{t\}\}$. Otherwise, L chooses between a split, merge, or stay move on Π_t, using proposals calibrated for the model $p_{t-1}(\Pi_{t-1}; y_{1:t-1})$, rather than the model calibrated to p_t used by the K kernel. In particular, upon receiving Π_t, L finds the cluster $\tilde{C} \in \Pi_t$ containing the point t, and sets $\tilde{C}' = \tilde{C} \setminus \{t\}$ and $\Pi_{t-1}^1 = \Pi_t \setminus \{\tilde{C}\} \cup \{\tilde{C}'\}$. L then uses the steps described in the

preceding paragraphs to decide whether to split cluster $\tilde{C}'$ into two new clusters, merge $\tilde{C}'$ with another cluster in Π^1_{t-1} n $f\tilde{C}'g$, or output $\Pi_{t-1} = \Pi^1_{t-1}$. The above steps are modified in three ways when used by the L proposal to choose to split, merge, or keep $\tilde{C}'$: (1) each score computed in the K kernel using $p_t(\ ; y_{1:t})$ is computed in L using $p_{t-1}(\ ; y_{1:t-1})$, (2) Q^K_{split} and h^K are replaced by Q^L_{split} and h^L, and (3) while split moves in the K kernel cannot split clusters initially containing only 2 points, because the K kernel may not perform a split move which results in a partition in which t is in a singleton cluster, the L kernel may split clusters containing 2 points.

The manner in which the K and L kernel constrain each others' random choices. In addition to proposing Π_t from Π_{t-1}, the K proposal must also output a specification of the value of every random choice the L kernel makes, such that if the L kernel made those choices when given Π_t, it would output the same Π_{t-1} K started with. Likewise, L must output a specification of every choice made by K. Some of the random choices made by L can be constrained using the choices K makes to choose Π_t, and vice versa: when K performs a split, L must perform the opposite merge, etc. There are other there are other choices made by L which it is unclear how to constrain by the random choices K needs to make to compute Π_t, and likewise there are choices made by K which L cannot constrain without sampling additional random values. Thus, K and L sample additional random choices solely for the sake of constraining each other.

The extra random choices K makes are as follows:

1. K samples all the randomness L would generate to sample and score its split moves, which are cannot be deterministically constrained by the choices K has made. If K did not choose to make a merge move, this means that L did not choose a split move, so all random choices L made to estimate the likely value of doing a split move need to be proposed by K. In this case, K can propose these choices from the exact same distribution L would use. If K did choose to make a merge move, L must have proposed a split move which inverts this merge move. This means that one of the N possible split moves proposed by the L kernel was consistent with the merge move K performed. K generates N 1 random split proposals for the L kernel to have made as the possible splits not selected as the chosen split move in order to estimate $\hat{s}_s$, and generates a final split move which is the exact opposite of the merge move it made (meaning the proposed C' and C'' are the clusters C'_* and C which K merged). To do this, K must sample auxiliary randomness u which Q^L_{split} may have generated in proposing the C' and C''; it does this by sampling u $h^L(C_* [\ C; C_*; C; y_{1:t-1}\ !\ )$.

The extra random choices L makes are as follows:

1. L samples all the randomness K would generate to sample and score its split moves. This is done symmetrically to how K generates the randomness needed for L's split-related proposals.
2. If L performed a split move, splitting cluster $\tilde{C}'$ into clusters $C'; C''$, then the locally-optimal datapoint assignment at the start of the K kernel may have initially placed t either into C' or C''; L must choose one of these options, and does so such that the probability of choosing C' and C'' are each proportional to the joint PDF of the resulting partition after putting t into the chosen cluster.

Given these additional random choices, L can constrain all the random choices K made for the sake of generating Π_t, and L can also constrain the additional random choices K made for the sake of constraining L by filling in the values these choices would need to take for L to output the constraint on K consistent with the choices it actually made during its execution. K constrains the choices in L similarly. (In the case where K added t to a singleton cluster, and L deterministically inverted this, the only random choice from K which L has to fill in is the fact that K chose to put it in its own cluster.)