SMCP^3: Sequential Monte Carlo 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

This paper introduces SMCP^3, a method for automatically implementing custom sequential Monte Carlo samplers for inference in probabilistic programs. Unlike particle filters and resample-move SMC (Gilks and Berzuini, 2001), SMCP^3 algorithms can improve the quality of samples and weights using pairs of Markov proposal kernels that are also specified by probabilistic programs. Unlike Del Moral et al. (2006b), these proposals can themselves be complex probabilistic computations that generate auxiliary variables, apply deterministic transformations, and lack tractable marginal densities. This paper also contributes an efficient implementation in Gen[1] that eliminates the need to manually derive incremental importance weights. SMCP^3 thus simultaneously expands the design space that can be explored by SMC practitioners and reduces the implementation effort. SMCP^3 is illustrated using applications to 3D object tracking, state-space modeling, and data clustering, showing that SMCP^3 methods can simultaneously improve the quality and reduce the cost of marginal likelihood estimation and posterior inference.

1 INTRODUCTION

This paper introduces SMCP^3, a probabilistic programming framework for custom sequential Monte Carlo (SMC) inference. Compared to modern SMC frameworks (Cusumano-Towner et al., 2018; Dai et al., 2022; Del Moral et al., 2006b; Gilks and Berzuini, 2001), SMCP^3 simultaneously expands the algorithm design space and reduces the implementation effort, making it easier for practitioners to efficiently improve the quality of samples and particle weights. SMCP^3 algorithms work by applying pairs of Markov proposal kernels to weighted particles. Unlike Del Moral et al. (2006b), these proposals can themselves be complex probabilistic programs that generate auxiliary variables, apply deterministic transformations, and lack tractable marginal densities. This paper describes a Gen implementation of SMCP^3 that automatically and efficiently calculates incremental importance weights (Sec. 4 and Appx. E). Redundant computation is avoided via static analysis, and Jacobian corrections are calculated via automatic differentiation.

SMCP^3 is illustrated using applications to 3D object tracking, state-space modeling, and data clustering. Expeirments show that on both synthetic and real-world datasets, SMCP^3 methods can simultaneously improve the quality and reduce the cost of marginal likelihood estimation and posterior inference, relative to strong SMC baselines. For example, for online state estimation, we show that an SMCP^3 algorithm can use a gradient-based Markov proposal to improve sample quality relative to particle filtering, and also use a simple backward Markov kernel to improve importance weights relative to resample-move SMC. For online data-clustering, we show that an SMCP^3 algorithm can improve over a locally-optimal SMC baseline, by splitting or merging clusters in response to each new datapoint. We also show that a pair of grid-based Markov kernels can simultaneously improve runtime performance and accuracy for 3D object tracking with a non-differentiable likelihood.

Contributions. This paper contributes:

1. The SMCP^3 mathematical framework (Section 3), which shows how to compute valid weights when proposals are general probabilistic programs that may not admit tractable marginal densities.
2. An method for efficiently automating custom SMCP^3 algorithms in probabilistic programming systems, and an implementation in Gen (Section 4).
3. Example SMCP^3 algorithms for state space modeling and data clustering, which exploit the new degrees of freedom in SMCP^3 and outperform strong resample-move and particle filter baselines (Section 5).

[1]`https://github.com/probcomp/GenSMCP3.jl`

Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206.

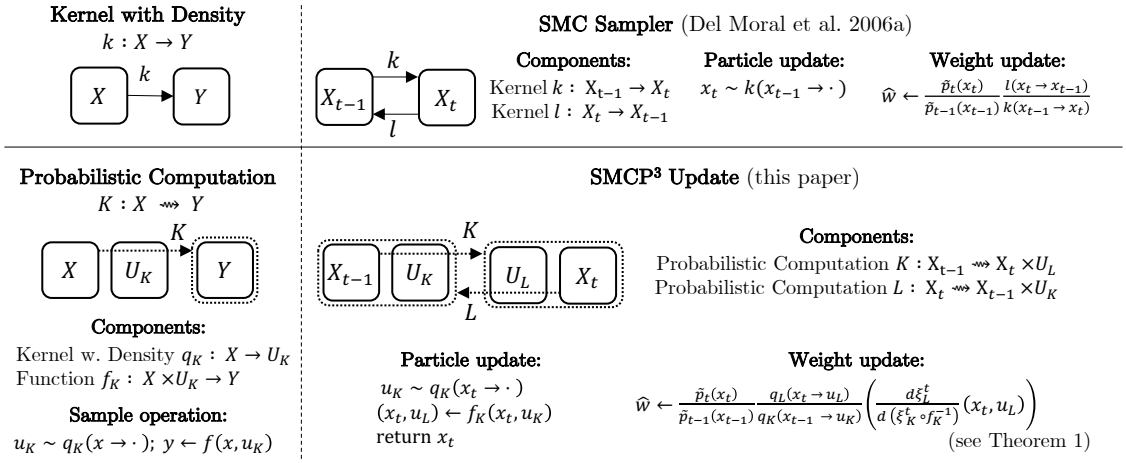

Figure 1: **Left:** In most treatments of SMC, proposals are assumed to have probability densities $k(x \to y)$ that can be evaluated exactly. To support more general proposals, we formalize SMCP^3 in terms of *probabilistic computations* $K : X \rightsquigarrow Y$, which make proposals in two steps: they first sample a random value u_K, then apply a deterministic function f_K to get y. Importantly, only the density of u_K must be available, not the marginal density of y. **Right:** SMCP^3 generalizes Del Moral et al. (2006a), by supporting K and L proposals specified as probabilistic computations, rather than as probability densities. In Section 4, we explain how any probabilistic program can be seen as a probabilistic computation, and show how to automate the calculation of the weight $\hat{w}$ when $\tilde{P}$, K and L are given as probabilistic programs.

2 BACKGROUND

Consider a sequence $(\tilde{P}_t)_{t \in \{1,\ldots,T\}}$ of unnormalized target distributions, defined over spaces X_t, and write P_t for the normalized targets $\frac{1}{\int \tilde{P}_t(dx)} \tilde{P}_t$. The entire sequence may be of intrinsic interest (corresponding, e.g., to evolving posteriors of an object's position as we receive sensor data), or it may be hand-designed to form a "bridge" of increasingly difficult subproblems, culminating in the true target P_T.

Example 2.1 (State-space models)**.** In *state-space models*, $X_t = Z^t$ for some state space Z, and $\tilde{P}_t$ are unnormalized filtering posteriors, with densities $\tilde{p}_t(z_{1:t}) = p_{init}(z_1) p_{obs}(y_1 \mid z_1) \prod_{i=2}^{t} p_{dyn}(z_i \mid z_{1:i-1}) p_{obs}(y_i \mid z_i)$ for some fixed sequence of observations $y_{1:t}$.

Example 2.2 (Data annealing)**.** In Bayesian data analysis, if we assume a dataset is drawn iid from $p_{obs}(y_i \mid \theta)$ for some latent $\theta \in \Theta$, we can take $X_t = \Theta$ for all t, and set target densities $\tilde{p}_t(\theta) = p_{prior}(\theta) \prod_{i=1}^{t} p_{obs}(y_i \mid \theta)$. Then the normalized target P_t is the posterior on θ given only the first t datapoints.

The goal of SMC is to develop particle approximations $\{(x_t^i, w_t^i)\}_{i=1}^N$ to each $\tilde{P}_t$, satisfying *proper weighting*.

Definition 1. A particle (x, w) is *properly weighted* for $\tilde{P}$ if $\mathbb{E}[w f(x)] = \int f(x) \tilde{P}(dx)$ for any integrable f.

SMC successively approximates each target, using the particles for one target as starting points to form particles for the next. After *initializing* $\{(x_1^i, w_1^i)\}_{i=1}^N$ via importance sampling targeting $\tilde{P}_1$, SMC alternates between *resampling* (selecting promising particles to serve as the basis for future inferences) and *updating* (modifying each particle independently to approximate the next target).

Example 2.3 (Bootstrap filter)**.** Consider a state-space model as in Example 2.1. The *bootstrap particle filter* updates a particle $(z_{1:t-1}, w)$ to $(z_{1:t}, w')$ by sampling $z_t \sim p_{dyn}(\cdot \mid z_{1:t-1})$ and computing $w' = w \cdot p_{obs}(y_t \mid z_t)$.

Example 2.4 (Resample-Move)**.** The *resample-move* filter (Chopin, 2002; Gilks and Berzuini, 2001) first runs MCMC moves (with invariant distribution P_{t-1}) on the previous state $z_{1:t-1}$, then runs a bootstrap update, with $z_t \sim p_{dyn}(\cdot \mid z_{1:t-1})$, and $w' = w \cdot p_{obs}(y_t \mid z_t)$.

Two key criteria often guide the design of SMC algorithms:

- **Sample quality.** How close are the samples x_t^i, in distribution, to the normalized targets P_t?
- **Weight quality.** How precisely do the weights measure the quality of the samples?

These goals can sometimes be in conflict. For example, because the bootstrap filter proposes samples according to the model's dynamics, without regard for the observed data y_t, its *sample quality* can be poor. The MCMC moves in the resample-move algorithm can improve sample quality, but these improvements are not reflected in the updated weights: the observations $y_{1:t-1}$ are not rescored and therefore the MCMC gets no "credit" for improving the existing trajectory. This poor *weight quality* can lead to less accurate importance sampling estimates of marginal likelihoods and posterior expectations. It also reduces the value of resampling, and thus can reduce end-to-end sample quality.

In this paper, we automate a broad family of SMC updates that help to navigate this tradeoff, to achieve good sample quality *and* accurate importance weights. Following Del Moral et al. (2006a), each SMC update is parameterized by two proposals, called K and L (Fig. 1, top):

- The proposal $K(x_{t-1} \to x_t)$ aims to propose a value x_t approximately distributed according to P_t, using a particle x_{t-1} from the existing collection to inform its proposal. Improving K improves the *sample quality*.
- The proposal $L(x_t \to x_{t-1})$ is used only to improve *weight quality*. Given x_t, the ideal L proposal assigns high probability to values of x_{t-1} which are likely to have been input to K, given that it generated x_t.

In theory, by carefully designing K and L proposals, practitioners can build arbitrarily accurate SMC algorithms. But in prior work, K and L must be chosen so their density ratio can be computed exactly, greatly restricting the design space. Furthermore, experimenting with K and L proposals can be tedious and error-prone: each change means new code for sampling and for computing weights, the soundness of which can be hard to unit-test. This paper presents (1) a new SMC algorithm that supports a broader class of K and L proposals, and (2) a probabilistic programming technique for automating its correct implementation, for K and L proposals expressed as probabilistic programs.

3 THE SMCP^3 ALGORITHM

Alg. 1 presents SMCP^3, our variant of SMC.[2] The key novelty in Alg. 1 is in how particle updates are implemented and how incremental importance weights are computed. Ultimately, we wish to accept user-specified probabilistic programs as K and L proposals, 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, bottom), which may simulate many random numbers, then apply deterministic post-processing to generate an update.

Definition 2. A *probabilistic computation* $K : X \rightsquigarrow Y$ between spaces X and Y is a tuple (U_K, Q_K, f_K), where:

- U_K is a space of auxiliary randomness,
- Q_K is a probability kernel taking an input $x \in X$ and outputting a sample $u_K \in U_K$, with density q_K, and
- $f_K : X \times U_K \to Y$ is a deterministic function.[3]

[2]To simplify the presentation, Alg. 1 uses multinomial resampling at every time step (L8-9). SMCP^3 also works with other popular resampling strategies (Douc and Cappé, 2005), as well as adaptive resampling policies based on effective sample size.

[3]Throughout the paper, we suppress some measure-theoretic details; for example, we must fix a reference measure on U_K with respect to which q_K is computed, and f_K must be a measurable map. See Appendix A for a more formal exposition.

Algorithm 1 SMCP^3

Require: Sequence of target densities $\tilde{p}_t$
Require: Initial proposal Q_1 w/ density q_1
Require: Forward proposals $K_t : X_{t-1} \rightsquigarrow X_t \times U_{L_t}$
Require: Backward proposals $L_t : X_t \rightsquigarrow X_{t-1} \times U_{K_t}$
Require: Number of particles N
Ensure: $(x_t^i, w_t^i)_{i=1}^N$ properly weighted for $\tilde{P}_t$

▷ Initialize particles with importance sampling
for $i = 1, \ldots, N$ **do**
  $x_1^i \sim Q_1$
  $w_1^i \leftarrow \frac{\tilde{p}_1(x_1^i)}{q_1(x_1^i)}$
end for
▷ Main SMC loop
for $t = 2, \ldots, T$ **do**
  ▷ Draw ancestor indices
  $(a_t^i)_{i=1}^N \sim$ Categorical$([w_{t-1}^1, \ldots, w_{t-1}^N])$
  ▷ Compute average weight
  $W_{t-1} \leftarrow \frac{\sum_{i=1}^N w_{t-1}^i}{N}$
  for $i = 1, \ldots, N$ **do**
    $\tilde{x} \leftarrow x_{t-1}^{a_t^i}$
    ▷ Draw auxiliary randomness u_{K_t}
    $u_{K_t} \sim Q_{K_t}(\tilde{x} \to \cdot)$
    ▷ Apply deterministic transformation
    $(x_t^i, u_{L_t}) \leftarrow f_{K_t}(\tilde{x}, u_{K_t})$
    ▷ Compute density ratio
    $\hat{w}_t^i \leftarrow \frac{\tilde{p}_t(x_t^i) q_{L_t}(x_t^i \to u_{L_t})}{\tilde{p}_{t-1}(\tilde{x}) q_{K_t}(\tilde{x} \to u_{K_t})}$
    ▷ Factor in change-of-variables (cf. Thm 1)
    $\hat{w}_t^i \leftarrow \hat{w}_t^i \cdot \frac{d\xi_L^t}{d(\xi_K^t \circ f_{K_t}^{-1})}(x_t^i, u_{L_t})$
    ▷ Weight update
    $w_t^i \leftarrow W_{t-1} \hat{w}_t^i$
  end for
end for

To run a probabilistic computation, we sample $u \sim Q_K(x \to \cdot)$, then compute $y = f_K(x, u)$. Importantly, we do not assume we know the marginal density of y.

In SMCP^3, users specify updates by defining a *pair* of probabilistic computations, to serve as K and L proposals:

Definition 3 (SMCP^3 move)**.** An *SMCP^3 move* from $\tilde{P}_{t-1}$ to $\tilde{P}_t$ is a pair of probabilistic computations $K_t : X_{t-1} \rightsquigarrow X_t \times U_{L_t}$ and $L_t : X_t \rightsquigarrow X_{t-1} \times U_{K_t}$, satisfying:

- **Full support:** If $\tilde{p}_t(x_t) q_{L_t}(x_t \to u_{L_t}) > 0$, then letting $(x_{t-1}, u_{K_t}) = f_{L_t}(x_t, u_{L_t})$, we must have $\tilde{p}_{t-1}(x_{t-1}) q_{K_t}(x_{t-1} \to u_{K_t}) > 0$.
- **Invertibility:** If $\tilde{p}_{t-1}(x_{t-1}) q_{K_t}(x_{t-1} \to x_t) > 0$, then $f_{L_t}(f_{K_t}(x_{t-1}, u_K)) = (x_{t-1}, u_K)$.

We now unpack the intuition behind this definition.

K proposal. The role of K_t in Alg. 1 is to transform a particle x_{t-1} from the approximation of $\tilde{P}_{t-1}$ into a particle

x_t for the approximation of $\tilde{P}_t$. To do so, it samples u_{K_t} (L15) and then applies a deterministic transformation f_K to (x_{t-1}, u_{K_t}) (L17). This yields not just an updated particle x_t, but also an extra output u_{L_t}, in which the user can stash any auxiliary data useful for meeting the invertibility requirement from Def. 3. Intuitively, we need the extra output u_{L_t} (and the invertibility requirement itself) thanks to two difficulties in computing an updated *weight* for x_t:

1. In a traditional particle filter, the weight update requires computing the density of x_t under the proposal. But in our setting, many different random samples u_{K_t} could result in the same proposed x_t, and computing the proposal's density would require intractable marginalization over these many *ways* K_t could have generated x_t. If we use our extra output u_{L_t} to record *which* of those ways *was* used, we can avoid this marginalization.
2. A traditional particle filter's weight update formula is based partly on the fact that the new state x_t is an *extension* of the old state x_{t-1}; no information about x_{t-1} is forgotten during the update. In our setting, K_t is an arbitrary probabilistic computation, which may edit its input x_{t-1}. If it does, u_{L_t} can record information about x_{t-1} overwritten or lost during the update.

The *invertibility* requirement in Def. 3 enforces that u_{L_t} stores enough information about the update to address both these challenges, enabling deterministic recovery (via f_{L_t}) of the previous state x_{t-1} and the randomness u_{K_t} of K_t.

L **proposal.** Unlike K_t, which is used to generate proposals, L_t is never run forward by Alg. 1; it is used only to inform the weight computation. The overall goal of L_t is to "guess" how K_t generated a given x_t: it outputs a hypothesized previous particle x_{t-1}, and hypothesized auxiliary randomness u_{K_t}. To do so, it samples $u_{L_t} \sim Q_{L_t}(x_t \to \cdot)$ and computes $(x_{t-1}, u_{K_t}) = f_{L_t}(x_t, u_{L_t})$.

The *full support* requirement in Def. 3 enforces that L_t can only make valid guesses: the x_{t-1} it guesses must be in the support of the previous target $\tilde{P}_{t-1}$, and the u_{K_t} it guesses must be within the support of $Q_{K_t}(x_{t-1} \to \cdot)$. Because we can feed any x_t in the support of $\tilde{P}_t$ *into* L_t, the requirement also enforces that K_t must have *some* way of proposing *any* x_t in the new target's support.

Computing weights. Given an SMCP^3 move, we can use K_t to propose a new particle value x_t (L14-17), but we still to compute a new weight w_t. The key desideratum is that (x_t, w_t) be *properly weighted* (Def. 1) for $\tilde{P}_t$.

The weight that Alg. 1 computes is the product of a density ratio (L19) with a *change-of-variables* correction (L21). The calculation is based on combining two ideas:

1. We choose to view the pair (x_t, u_{L_t}) as a proposal for an extended target distribution, $\tilde{p}_t(x_t)q_{L_t}(x_t \to u_{L_t})$. Marginalizing u_{L_t} yields the original target $\tilde{p}_t$, so if $((x_t, u_{L_t}), w_t)$ is properly weighted for the extended target, then (x_t, w_t) will be properly weighted for the original target. This explains the numerator in L19.
2. We compute a change-of-variables correction to account for the transformation, by f_{K_t}, of samples (x_{t-1}, u_{K_t}) into samples (x_t, u_{L_t}). The denominator in L19 gives the proposal density *before* the transformation, and then L21 multiplies in the correction.

Thanks to invertibility (Def. 3), f_{K_t} behaves like a bijection, enabling the application of standard techniques for computing change-of-variables corrections. For example, when all model and proposal variables are continuous and f_{K_t} is differentiable, the correction is the absolute value of the determinant of the Jacobian matrix of f_{K_t}. More generally, the correction is a *Radon-Nikodym derivative*:

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 $u_K \sim Q_K(x \to \cdot)$, and $(x', u_L) = f_K(x, u_K)$, the pair $(x', \hat{w} \cdot w)$ is properly weighted for $\tilde{P}_t$, where

$$\hat{w} = \frac{\tilde{p}_t(x')q_L(x' \to u_L)}{\tilde{p}_{t-1}(x)q_K(x \to u_K)} \cdot \frac{d\xi_L^t}{d(\xi_K^t \circ f_K^{-1})}(x', u_L),$$

and ξ_L^t and ξ_K^t are reference measures defined in Appx. A.

The correction factor can be calculated in very general settings (Lew et al., 2021b; Radul and Alexeev, 2021), including whenever f_K can be expressed in a Turing-complete language with piecewise-differentiable primitives (Huot et al., 2023). In Sec. 4, we show how to automate it when models and moves are given as Gen probabilistic programs.

Practical proposal design guidelines. K_t should be designed so that the x_t it computes is approximately distributed according to the new target P_t, assuming the input value x_{t-1} is distributed according to P_{t-1}.Thanks to invertibility, once K_t is fixed, the only degree of freedom that remains in designing L_t is to choose the proposal Q_{L_t}. A reasonable strategy is to approximate the *locally optimal* proposal, as characterized by the following proposition:

Proposition 1. Consider the distribution of (x_t, u_{L_t}) when K_t is run on $x_{t-1} \sim p_{t-1}$. The conditional distribution of u_{L_t} given x_t is the *locally optimal* choice of Q_{L_t}, minimizing the variance of the incremental weight $\hat{w}$.[4]

In practice, the runtime of K_t and L_t will also be important considerations, as using faster proposals enables higher particle counts. For example SMCP^3 proposals, see Sec. 5.

Convergence of SMCP^3. SMCP^3 algorithms can be formulated as Feynman-Kac models, so we can use standard convergence arguments (Chopin et al., 2020).[5]

[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 $w^t = \hat{w} \cdot W_{t-1}$.

[5]Prop. 2 sometimes holds even when (K_t, L_t) are chosen

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

(a) Gen implementation of a simple dynamic model.

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

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

Figure 2: Gen implementation of the example in Sec. 5.1.

Proposition 2 (Central Limit Theorem)**.** If $\tilde{P}_t$, K_t, L_t are such that the incremental weights $\hat{w}_i^t$ in Alg. 1 are bounded above, then for any continuous, bounded function $\varphi : X_t \to \mathbb{R}$, there exists σ s.t.

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

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

4 AUTOMATING SMCP^3

In this section, we show how to *automate* correct implementations of SMCP^3, when the target distributions and particle updates are specified as probabilistic programs.

Gen programs. We work with the PPL Gen (Cusumano-Towner et al., 2019), in which a probabilistic program is an ordinary (deterministic) Julia function, augmented with the syntactic construct $\{\texttt{name}\} \sim \texttt{distribution}$. This statement causes Gen to sample a random value from a distribution (e.g., $\texttt{normal}(0, 4)$), 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 $\texttt{Trace}(\texttt{"x1"} \Rightarrow 0.4, \texttt{"y1"} \Rightarrow 1.1, \texttt{"x2"} \Rightarrow 0.2, \texttt{"y2"} \Rightarrow 0.3)$. The values in the trace will vary from run to run, and 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). Appx. A reviews the standard measure-theoretic machinery used to reason rigorously about distributions over the space $\mathbb{T}$ of program traces. The upshot is that the *density* of a trace under a probabilistic program can be computed by multiplying the conditional densities of each primitive distribution encountered while generating the trace—just as we do when computing a joint density for a Bayes net.

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 $\tilde{P}^{y_t}(t \to \cdot)$

$u_{K_t} \leftarrow \texttt{SAMPLE-TRACE}(K_t, x)$
$((x', u_{L_t}), \hat{J}) \leftarrow \texttt{AD}(\texttt{COMPUTE-RETVAL}(K_t, x, u_{K_t}))$
$\log q_{K_t} \leftarrow \texttt{EVALUATE-LOGPDF}(K_t, x, u_{K_t})$
$\log \tilde{p}_{t-1} \leftarrow \texttt{EVALUATE-LOGPDF}(P, t-1, x \oplus y_{t-1})$
$\log \tilde{p}_t \leftarrow \texttt{EVALUATE-LOGPDF}(P, t, x' \oplus y_t)$
$\log q_{L_t} \leftarrow \texttt{EVALUATE-LOGPDF}(L_t, x', u_{L_t})$
$\log r \leftarrow \log \tilde{p}_t - \log \tilde{p}_{t-1} + \log q_{L_t} - \log q_{K_t}$
$\log w' \leftarrow \log w + \log r + \log |\det \hat{J}|$
return $(x', \log w')$

Gen programs as probabilistic computations. A Gen probabilistic program taking arguments in space A and returning values in space B implements a probabilistic computation $P : A \rightsquigarrow B$. The space U_P of auxiliary randomness is $\mathbb{T}$, the space of execution traces; the auxiliary distribution $Q_P(a \to \cdot)$ is the distribution over traces induced by the program; and $f_P : A \times \mathbb{T} \to B$ 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. $\texttt{SAMPLE-TRACE}(P, a)$: Sample $\tau \sim Q_P(a \to \cdot)$.
2. $\texttt{EVALUATE-LOGPDF}(P, a, \tau)$: Return $\log q_P(a \to \tau)$.
3. $\texttt{COMPUTE-RETVAL}(P, a, \tau)$: Return $f_P(a, \tau)$.

Defining sequences of unnormalized targets. Given an argument a, a probabilistic program P defines a normalized probability distribution over traces containing many named variables. Just as a joint density $p(x, y)$ can be recast as an *unnormalized* density $\tilde{p}(x) = p(x, \mathbf{y})$ for fixed

adaptively based on properties of the current particles (Beskos et al., 2016; Fearnhead and Taylor, 2013). But, as in standard SMC, such updates may invalidate proper weighting for finite N.

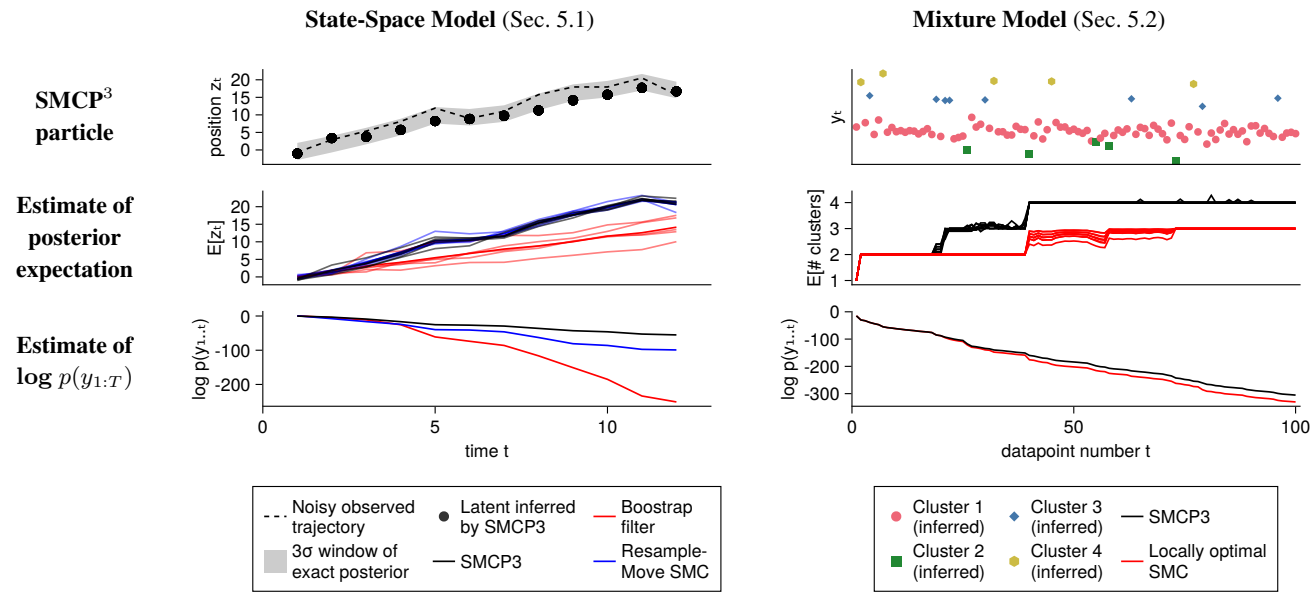

Figure 3: Illustrative results on synthetic data. **Top:** A single inferred posterior sample, in two models. In the state-space model, a true position is tracked from noisy observations over time. In the mixture model, clusterings are inferred for increasing subsets of the data. **Middle:** Estimates of posterior expectations at each step, formed from particle collections. We run each algorithm 5 times. **Bottom:** Log marginal likelihood estimates at each inference step; higher is better.

data $\mathbf{y}$, we can specify unnormalized targets in Gen by fixing a *trace* $\mathbf{y} \in \mathbb{T}$ of observations, mapping some of the names at which P can sample to observed values. We define $\tilde{P}^{\mathbf{y}}$ as the unnormalized kernel with density $\tilde{p}^{\mathbf{y}}(a \to \tau) = q_P(a \to \tau \oplus \mathbf{y})$, where $\oplus$ *merges* two traces with disjoint sets of names. (If τ shares names with $\mathbf{y}$, we define $\tilde{p}^{\mathbf{y}}(\tau)$ to be 0.) A user defines a *sequence* of unnormalized targets 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. This yields the sequence $(\tilde{P}^{\mathbf{y}_t}(t \to \cdot))_{t=1}^T$.

Specifying K and L. For $2 \leq t \leq T$, the user specifies $K_t : \mathbb{T} \rightsquigarrow \mathbb{T} \times \mathbb{T}$ and $L_t : \mathbb{T} \rightsquigarrow \mathbb{T} \times \mathbb{T}$, as probabilistic programs that accept model traces x as input and return model and proposal traces as output (see Fig. 2b).[6] For any user-specified K and L, the two conditions from Def. 3 can be automatically fuzz-tested; for details, see Appx. E.2.

Automating SMCP^3. Given a model probabilistic program P, observations $\mathbf{y}_{t-1}$ and $\mathbf{y}_t$, and K_t and L_t, Alg. 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),[7] and then computes the particle weight update (L7-8). To compute the incremental weight, step (4) must compute the change-of-variables correction from Thm. 1. Thm. 2 shows that this factor can be computed by treating f_{K_t} as a function of the continuous values in (x_{t-1}, u_{K_t}) (ignoring the discrete values in this pair), and computing the determinant of its Jacobian ($\hat{J}$, in Alg. 2).[8]

Theorem 2 (Informal). Let (K_t, L_t) be an SMCP^3 move such that f_{K_t} satisfies the differentiability-related conditions in Appx. D. Let $x_{t-1} \sim P_{t-1}$ and $u_{K_t} \sim Q_{K_t}(x_{t-1} \to \cdot)$. Let $x^c_{t-1} +\!\!+ u^c_{K_t} \in \mathbb{R}^D$ denote the concatenation of all the continuous values in x_{t-1} and u_{K_t}. Let $g : \mathbb{R}^D \to X_{t-1} \times U_{K_t}$ be the function which replaces the continuous values in (x_{t-1}, u_{K_t}) with the values in the vector input to it, such that $g(x^c_{t-1} +\!\!+ u^c_{K_t}) = (x_{t-1}, u_{K_t})$. Then with probability 1, there exists a neighborhood $\mathcal{O}_K$ of $x^c_{t-1} +\!\!+ u^c_{K_t}$ such that every element (x'_t, u'_L) of $(f_{K_t} \circ g)(\mathcal{O}_K)$ contains D continuous values, and if $(x_t, u_{L_t}) = f_{K_t}(x_{t-1}, u_{K_t})$,

$$\frac{d\xi^t_{L_t}}{d(\xi^t_{K_t} \circ f_{K_t}^{-1})}(x_t, u_{L_t}) = |\det J(h \circ f_{K_t} \circ g)(x^c_{t-1} +\!\!+ u^c_{K_t})|$$

where J takes the Jacobian, $h : (f_{K_t} \circ g)(\mathcal{O}_K) \to \mathbb{R}^D$ is a function extracting all the continuous values from any pair (x'_t, u'_L) into a vector, and $\frac{d\xi^t_{L_t}}{d(\xi^t_{K_t} \circ f_{K_t}^{-1})}$ is as in Thm. 1.

[6] Both models and proposals are probabilistic programs, so Sec. 3's state spaces X_t and auxiliary spaces U_K, U_L are all $\mathbb{T}$.

[7] Alg. 2 separately computes $\log \tilde{p}_{t-1}$ and $\log \tilde{p}_t$. In Gen, we directly compute their *difference*, for asymptotic speedups in many cases (Cusumano-Towner et al., 2019); see Appendix E.3.

[8] To compute $\hat{J}$, our implementation uses forward-mode AD: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', u_{L_t}, to fill out the Jacobian matrix $\hat{J}$. (We delete the dual components before applying further operations to x' and u_{L_t} on L5-6.)

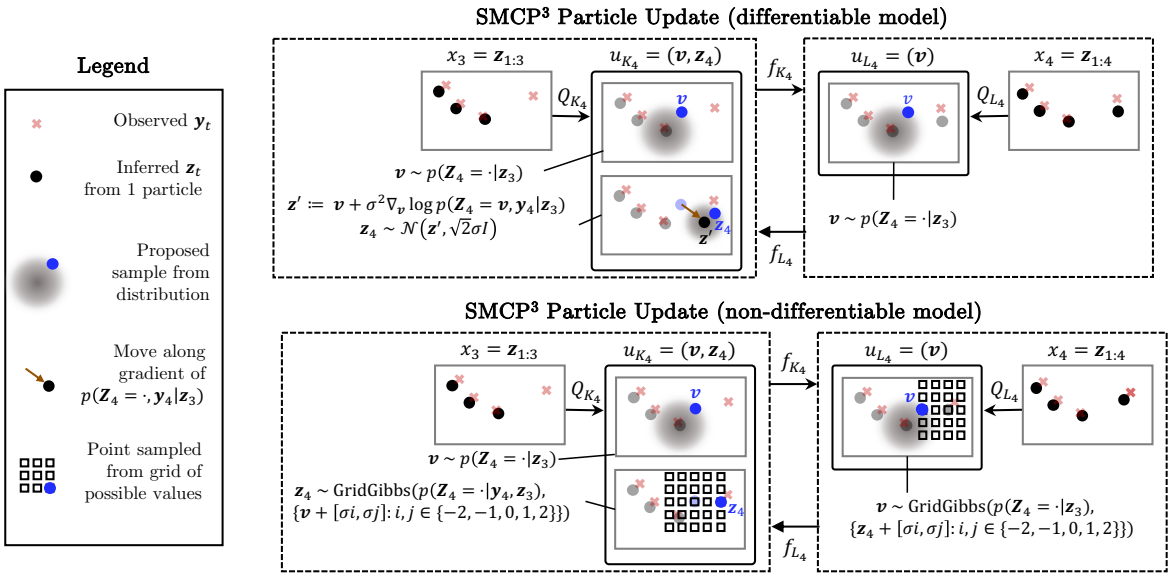

Figure 4: **Top:** A 2D illustration of the SMCP^3 move from Fig. 2b, for online Bayesian state estimation. **Bottom:** A variant of the move for use in models with non-differentiable probability densities, like that of Fig. 6. We use the notation $p(\mathbf{Z}_4 = \cdot, \mathbf{y}_4 | \mathbf{z}_3)$ to indicate the joint density of $(\mathbf{z}_4, \mathbf{y}_4)$ given $\mathbf{z}_3$, and $p(\mathbf{Z}_4 = \cdot | \mathbf{z}_3)$ to indicate the density of $\mathbf{z}_4$ given $\mathbf{z}_3$.

5 EXAMPLES

We now demonstrate how probabilistic program proposals can be used to improve inference, relative to baselines that use simpler proposals with closed-form marginal densities.

When comparing algorithms for the same inference problem, we report log marginal likelihood estimates as measures of relative inference quality. In some applications (e.g., particle MCMC (Andrieu et al., 2010)), these estimates are of direct interest, but even if we care only about accurate posterior inference, higher log marginal likelihoods suggest lower KL divergences between the true and approximate posteriors (Lew et al., 2022, Thm. 4).[9] In this section we run each algorithm with equal particle counts; Appx. G shows similar results controlling for runtime.

5.1 Online Bayesian State Estimation

Model. We first study the Gaussian state-space model from Fig. 2a. We take $\mathbf{z}_0 = \mathbf{0} \in \mathbb{R}^d$, and for $t > 0$, set $\mathbf{z}_t \sim \mathcal{N}(\mathbf{z}_{t-1}, I)$ and $\mathbf{y}_t \sim \mathcal{N}(\mathbf{z}_t, I)$. We define our target distributions over the spaces $X_t = \mathbb{R}^{dt}$, where each $x_t \in X_t$ is a trajectory $\mathbf{z}_{1:t}$. The unnormalized target distributions are proportional to the filtering posteriors: $\tilde{p}_t(\mathbf{z}_{1:t}) = p(\mathbf{z}_{1:t}) p(\mathbf{y}_{1:t} \mid \mathbf{z}_{1:t}) \propto p(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t})$.

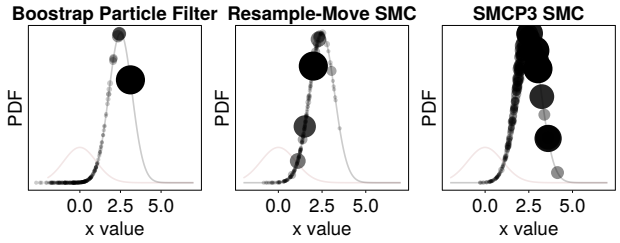

Figure 5: SMCP^3 vs baselines for online state estimation, in the model from Figure 2a. Weighted particles generated by SMC at $t = 1$, given $\mathbf{z}_1 = 5$. Pink curve: PDF of $P(\mathbf{Z}_1)$. Grey curve: PDF of the target $P(\mathbf{Z}_1 | y_1)$.

SMCP^3 algorithm. We set K_t and L_t as in Fig. 2b. As illustrated in Fig. 4, K_t extends $\mathbf{z}_{1:t-1}$ with a new value $\mathbf{z}_t$, generated by performing an unadjusted Langevin ascent move from a random initial position $\mathbf{v} \sim \mathcal{N}(\mathbf{z}_{t-1}, I)$. Our simple choice of L_t ignores $\mathbf{z}_t$ when proposing $\mathbf{v}$, generating it from $\mathcal{N}(\mathbf{z}_{t-1}, I)$. Specifically, we have $u_{K_t} = (\mathbf{v}, \mathbf{z}_t)$, and we take $q_{K_t}(\mathbf{z}_{1:t-1} \to (\mathbf{v}, \mathbf{z}_t)) = \mathcal{N}(\mathbf{v}; \mathbf{z}_{t-1}, I) \mathcal{N}(\mathbf{z}_t; \mathbf{z}', \sqrt{2}\sigma I)$ where $\mathbf{z}' = \mathbf{v} + \sigma^2 \nabla_{\mathbf{v}} p(\mathbf{Z}_t = \mathbf{v}, \mathbf{y}_t | \mathbf{z}_{t-1})$, σ is the step-size for the Langevin move, and $\mathcal{N}$ denotes the density of a Gaussian. We then have $f_{K_t}(\mathbf{z}_{1:t-1}, (\mathbf{v}, \mathbf{z}_t)) = (\mathbf{z}_{1:t}, \mathbf{v})$. u_{L_t} is just the value $\mathbf{v}$, with distribution $q_{L_t}(\mathbf{z}_{1:t} \to \mathbf{v}) = \mathcal{N}(\mathbf{v}; \mathbf{z}_{t-1}, I)$. The map $f_{L_t}(\mathbf{z}_{1:t}, \mathbf{v}) = (\mathbf{z}_{1:t-1}, (\mathbf{v}, \mathbf{z}_t))$.

[9] Consider a generative model $p(x, y)$ with observed data $\mathbf{y}$. Any SMC algorithm will produce an unbiased estimate of the marginal likelihood $p(\mathbf{y})$, but by Jensen's inequality, the logarithm of this estimate is a *biased* estimate of $\log p(\mathbf{y})$, and the bias is always negative. The magnitude of this bias is bounded below by the divergence $D_{KL}(p(x \mid \mathbf{y}) || q_{\text{ALG}}(x; \mathbf{y}))$, where q_{ALG} is the marginal distribution of one resampled particle from the SMC algorithm in question (Lew et al., 2022). Algorithms with higher expected log marginal likelihood estimates have smaller negative bias, and in many cases, will have lower KL divergence to the true posterior.

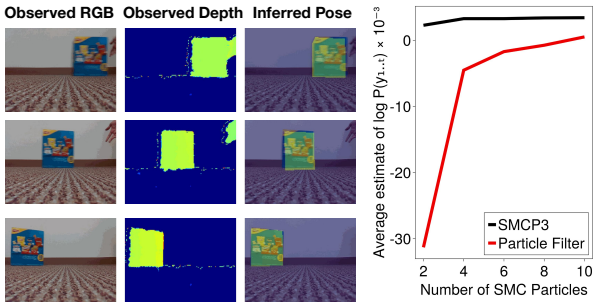

Figure 6: Object tracking from depth video in a non-differentiable state-space model.

Baselines. We compare to two baselines: a standard bootstrap particle filter (Ex. 2.3), and a resample-move SMC algorithm (Ex. 2.4), with Metropolis-adjusted Langevin ascent (MALA) rejuvenation on $\mathbf{x}_t$ after each step.

Results. Fig. 3 illustrates inference on an example sequence of 1-D observations, and Tab. 1 gives log marginal likelihood estimates for a synthetic 100-D dataset. SMCP^3 outperforms both the bootstrap filter and the stronger resample-move baseline; the 1-D illustration in Fig. 5 sheds light on why. The bootstrap 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 uses an identical Langevin step to move the particles, but accounts for this move in the weights, so that after resampling the particle cloud will better reflect the new posterior. SMCP^3 also skips the Metropolis-Hastings accept/reject step used in MALA, resulting in more particles moving.

Non-differentiable variant. In Figs. 4 and 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 (Gothoskar et al., 2021). Instead of a randomly initialized Langevin move, the K kernel samples a proposed point from a coarse grid centered at a random location, based on relative likelihoods at each grid point. As in the differentiable model, this improves over the bootstrap particle filter; see Appendix F.1.2 for details.

5.2 Online Bayesian Data Analysis

Model. We next consider a sequence of collapsed Dirichlet process mixture models (DPMMs), each incorporating an additional data-point. The t^{th} model places a $CRP(\{1,\ldots,t\})$ prior over partitions Π_t of $\{1,\ldots,t\}$, and for each cluster $C \in \Pi_t$, generates data $(y_i)_{i\in C}$ jointly from an exchangeable likelihood $F(\mathbf{y})$.[10] We observe $y_{1:t}$, and inference targets the posterior over Π_t given $y_{1:t}$.

[10]We use different likelihoods F, to model numerical data, and strings in a Medicare dataset; see Appx. F.2.1 for details.

	Est. of $\log P(\mathbf{y}_{1:T})$
State-space model (Sec. 5.1)	
100D trajectory (synthetic)	
Bootstrap PF	-4347.67 $\pm$ 83.21
Resample-Move SMC	-2828.29 $\pm$ 22.78
SMCP^3 ULA	**-2271.03 $\pm$ 10.86**
Mixture model (Sec. 5.2)	
Medicare data	
Locally Optimal SMC	-40851.15 $\pm$ 0.98
Resample-Move Split/Merge	-16898.79 $\pm$ 1117.31
SMCP^3 Split/Merge	**-13882.47 $\pm$ 0.24**

Table 1: Expected log marginal likelihood estimates returned by different algorithms; higher is better. We report the empirical mean across many runs, and the standard deviation of our estimate across repeated multiple-run trials.

SMCP^3 algorithm. Alg. 3 defines our proposal. K_t accepts as input a partition Π_{t-1} of the first $t-1$ datapoints and proposes a partition Π_t, incorporating the new point y_t. To do so, it first performs a "Gibbs" assignment of y_t to an existing cluster $C_* \in \Pi_{t-1}$, or to a new cluster; if a new cluster, K_t stops early and proposes $\Pi_t := \Pi_{t-1} \cup \{\{t\}\}$. Otherwise, K_t decides between splitting C_*, merging C_* with another cluster, or leaving C_* be. This choice is made based on a Monte Carlo estimate of the total probability of all states in which C_* is split; see Appx. F.2.2 for details. L_t identifies the cluster $C'_* \in \Pi_t$ 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 proposal to incorporate y_t into the clustering (without splitting or merging), and a resample-move version which adds split/merge MH moves.

Results. Tab. 1 reports log marginal likelihood estimates on 1k rows from the Medicare Hospital Compare dataset (Medicare, 2012). The locally optimal baseline greedily assigns points to clusters, which can lead it to get stuck in local modes. Both SMCP^3 and resample-move are able to escape local modes, but as in Sec. 5.1, SMCP^3 computes better weights after splits/merges, leading to improved log likelihood estimates.

6 RELATED WORK AND DISCUSSION

SMC. SMCP^3 provides automation for many SMC algorithms, including SMC samplers (Del Moral et al., 2006b), resample-move SMC (Gilks and Berzuini, 2001), and move-reweight SMC (Marques and Storvik, 2013). Unlike this prior work, SMCP^3 can be used to compute proper SMC weights for proposals that incorporate auxiliary variables (Fearnhead et al., 2010; Finke, 2015; Lew et al., 2022) and deterministic transformations. For inference over $\mathbb{R}^n$, Everitt et al. (2020) show a technique for incorporating deterministic logic into SMC updates, with Jacobian corrections similar to ours. SMCP^3 generalizes and extends their technique, adapting it to the more traditional SMC setting with K and L kernels, and automating its ap-

Algorithm 3 SMCP^3 K kernel from Sec. 5.2.

Require: Partition Π_{t-1} of $\{1, \ldots, t-1\}$, data $y_{1:t}$
Ensure: Partition Π_t of $\{1, \ldots, t\}$

procedure K($t, \Pi_{t-1}, y_{1:t}$)
  ▷ Incorporate y_t into an existing or new cluster.
  $\vec{\Phi} \leftarrow [\Pi_{t-1} \setminus \{c\} \cup \{c \cup \{t\}\} : c \in \Pi_{t-1}]$
  $\vec{\Phi} \leftarrow \vec{\Phi} +\!\!+ [\Pi_{t-1} \cup \{\{t\}\}]$ ▷ Append to $\vec{\Phi}$
  $j \sim$ Categorical($[p_t(\Pi, y_{1:t}) : \Pi \in \Phi]$)
  $\Pi_t^1 \leftarrow \vec{\Phi}[j]$; $c_t \leftarrow$ `ClusterContaining`(Π_t, t)
  ▷ Consider splitting or merging c_t.
  $\vec{\Pi} \leftarrow [\Pi_t^1]$ ▷ Vector of possible Π_t to output.
  $\vec{\Theta} \leftarrow [p_t(\Pi_t^1, y_{1:t})]$ ▷ Scores for elements of $\vec{\Pi}$.
  if $|c_t^1| > 1$ **then**
    ▷ Consider merges c_t of with other clusters
    for $c \in \Pi_t^1 \setminus \{c_t\}$ **do**
      ▷ Merge c and c_t.
      $\Pi_t^M \leftarrow \Pi_t^1 \setminus \{c, c_t\} \cup \{c \cup c_t\}$
      $\theta^M \leftarrow p_t(\Pi_t^M, y_{1:t})$
      $(\vec{\Pi}, \vec{\Theta}) \leftarrow (\vec{\Pi} +\!\!+ [\Pi_t^M], \vec{\Theta} +\!\!+ [\theta^M])$
    end for
  end if
  if $|c_t^1| > 2$ **then**
    ▷ `ProposeSplit` samples a post-split clustering Π_t^S, its score θ^S, and auxiliary choices u^S.
    $\Pi_t^S, \theta^S, u^S \sim$ `ProposeSplit`$(\Pi_t^1, y_{1:t})$
    $(\vec{\Pi}, \vec{\Theta}) \leftarrow (\vec{\Pi} +\!\!+ [\Pi_t^S], \vec{\Theta} +\!\!+ [\theta^S])$
  else
    $u^S \leftarrow$ `nil`
  end if
  $i \sim$ Categorical($\vec{\Theta}$); $\Pi_t \leftarrow \vec{\Pi}[i]$ ▷ Final clustering.
  $u_L \sim$ `GenerateUL`$(y_{1:t}, \Pi_t, \Pi_{t-1}, \Pi_t^S, u^S)$
  return (Π_t, u_L)
end procedure

plication when models and proposals are probabilistic programs. SMCP^3 moves could also be incorporated into the many algorithms that use SMC moves as building blocks, including extensions to SMC (Chopin et al., 2013; Kuntz et al., 2021; Lindsten et al., 2017) and pseudomarginal algorithms (Andrieu et al., 2010; Lindsten et al., 2014).

Probabilistic programming. Many PPLs support automated SMC for models specified as probabilistic programs (Cusumano-Towner et al., 2019; Ge et al., 2018; Goodman and Stuhlmüller, 2014; Lundén et al., 2021; Mansinghka et al., 2014, 2018; Milch et al., 2006; Paige and Wood, 2014; Ścibior et al., 2015; Wood et al., 2014). Some also have support for restricted classes of custom SMC with proposals defined as probabilistic programs (Bingham et al., 2019; Cusumano-Towner et al., 2018, 2019; Murray, 2013; Murray and Schön, 2018; Stites et al., 2021). These languages' restrictions prohibit proposals that sample auxiliary variables or deterministically transform samples. Stites et al. (2021) present combinators for defining samplers, including a `propose` combinator for custom SMC proposals. While these proposals can use auxiliary variables, they are ignored for weight computation, yielding sound but potentially high-variance weights (equivalent to a sub-optimal choice of L kernel in SMCP^3).

Involutive MCMC. SMCP^3 is inspired by automated involutive MCMC (Andrieu et al., 2020; Cusumano-Towner et al., 2020; Matheos et al., 2020; Neklyudov et al., 2020) (IMCMC), an analogously general framework for automated, custom MH with probabilistic program proposals. Indeed, the relationship between SMCP^3 and IMCMC is the same as that between SMC Samplers (Del Moral et al., 2006b) and ordinary MH (Marques and Storvik, 2013). When an SMC move leaves the target unchanged ($\tilde{P}_{t-1} = \tilde{P}_t$) and uses identical K and L proposals, an SMC Sampler's incremental weight is precisely the ordinary MH acceptance ratio with proposal $Q = K = L$, and SMCP^3's incremental weight is precisely the *involutive* MCMC acceptance ratio with proposal $Q = Q_K = Q_L$ and involution $f = f_K = f_L$. This connection clarifies one way in which SMC generalizes MCMC: many programs could serve as SMC K proposals that would *not* be valid MCMC proposals, since they are not reversible and so $L \neq K$.

Discussion. SMCP^3 gives inference algorithm designers a more flexible framework than previous formulations, while automating the implementation details. Initial experiments show that proposals using the new degrees of freedom that SMCP^3 offers can yield more accurate inference than strong baselines. One important area for future work is to improve speed through PPL compilation techniques (Cusumano-Towner et al., 2019; Huang et al., 2017; Lundén et al., 2022; Murray, 2020; Paige and Wood, 2014; Wu et al., 2016) and massively parallel hardware (Durham and Geweke, 2011; Lundén et al., 2022; Murray et al., 2016; Paige et al., 2014). It may also be possible to automatically tune the runtime and robustness of probabilistic program proposals, using data-dependent (Cusumano-Towner and Mansinghka, 2017) and model-averaged (Domke, 2021) estimators of SMC's accuracy, or recent methods for differentiating through SMC (Arya et al.; Corenflos et al., 2021; Lai et al., 2022; Lew et al., 2023; Maddison et al., 2017; Naesseth et al., 2018; Ścibior and Wood, 2021; Zhu et al., 2020).

Many questions about how best to design probabilistic proposals remain open. For example, when is it worthwhile to improve sample quality with a more complex K kernel, at the expense of making it harder for L to generate high-quality weights? We hope SMCP^3's Gen implementation will help researchers explore these questions, and help practitioners apply advanced SMC to complex modeling problems, by experimenting with richer probabilistic computations in their proposals (perhaps based on backtracking search, optimization, and model-based planning).

Acknowledgements

We have benefited from conversations with many friends and colleagues, including Mathieu Huot, Feras Saad, Marco Cusumano-Towner, Matthew Hoffman, Mirko Klukas, McCoy Becker, Cameron Freer, Eli Sennesh, and Jan-Willem van-de-Meent. We are also grateful to anonymous referees for very helpful feedback. Researchers from the MIT Probabilistic Computing Project were supported by NSF Graduate Research Fellowship (under Grant 1745302), DARPA (under the MCS and SAIL-ON programs), the Singapore DSTA, an OpenPhil AI fellowship, and philanthropic gifts from Google, an anonymous donor, and the Siegel Family Foundation. Researchers from UC Berkeley were supported by an AI2050 Senior Fellowship from Schmidt Futures, and an Open Philanthropy Foundation gift to the Center for Human-Compatible AI at Berkeley.

References

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.

Christophe Andrieu, Anthony Lee, and Sam Livingstone. A general perspective on the Metropolis-Hastings kernel. *arXiv preprint arXiv:2012.14881*, 2020.

Gaurav Arya, Moritz Schauer, Frank Schäfer, and Christopher Vincent Rackauckas. Automatic differentiation of programs with discrete randomness. In *Advances in Neural Information Processing Systems*.

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.

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.

Nicolas Chopin. A sequential particle filter method for static models. *Biometrika*, 89(3):539–552, 2002.

Nicolas Chopin, Pierre E Jacob, and Omiros Papaspiliopoulos. SMC^2: 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.

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

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.

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.

Marco Cusumano-Towner, Benjamin Bichsel, Timon Gehr, Martin Vechev, and Vikash K Mansinghka. Incremental inference for probabilistic programs. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 571–585, 2018.

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

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.

Chenguang Dai, Jeremy Heng, Pierre E Jacob, and Nick Whiteley. An invitation to sequential Monte Carlo samplers. *Journal of the American Statistical Association*, 117(539):1587–1600, 2022.

Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential Monte Carlo for Bayesian computation. *Bayesian Statistics*, 8, 2006a.

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, 2006b.

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

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.

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 Bastürk, Lennart Hoogerheide, Anne Opschoor, Herman K. van Dijk*, 29:17–34, 2011.

Richard G Everitt, Richard Culliford, Felipe Medina-Aguayo, and Daniel J Wilson. Sequential Monte Carlo with transformations. *Statistics and computing*, 30(3): 663–676, 2020.

Paul Fearnhead and Benjamin M Taylor. An adaptive sequential Monte Carlo sampler. *Bayesian analysis*, 8(2): 411–438, 2013.

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.

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

Hong Ge, Kai Xu, and Zoubin Ghahramani. Turing: a language for flexible probabilistic inference. In *International Conference on Artificial Intelligence and Statistics*, pages 1682–1690. PMLR, 2018.

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.

Noah D Goodman and Andreas Stuhlmüller. The Design and Implementation of Probabilistic Programming Languages. `http://dippl.org`, 2014. Accessed: 2023-2-23.

Nishad Gothoskar, Marco Cusumano-Towner, Ben Zinberg, Matin Ghavamizadeh, Falk Pollok, Austin Garrett, Josh Tenenbaum, Dan Gutfreund, and Vikash Mansinghka. 3DP3: 3D scene perception via probabilistic programming. *Advances in Neural Information Processing Systems*, 34:9600–9612, 2021.

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.

Mathieu Huot, Alexander K. Lew, Vikash K. Mansinghka, and Sam Staton. ωPAP spaces: Reasoning denotationally about higher-order, recursive probabilistic and differentiable programs. 2023. doi: 10.48550/arxiv.2302.10636. URL `https://arxiv.org/abs/2302.10636`.

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.

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

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.

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

Alexander K Lew, Ben Sherman, Marco F Cusumano-Towner, Michael Carbin, and Vikash K Mansinghka. On the automatic derivation of importance samplers from pairs of probabilistic programs. *Languages for Inference Workshop (LAFI)*, 2021b.

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 Artificial Intelligence*, 2022. URL `https://openreview.net/forum?id=BzMxEdIsqeq`.

Alexander K Lew, Mathieu Huot, Sam Staton, and Vikash K Mansinghka. ADEV: Sound automatic differentiation of expected values of probabilistic programs. *Proceedings of the ACM on Programming Languages*, 7 (POPL):121–153, 2023.

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

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.

Daniel Lundén, Johannes Borgström, and David Broman. Correctness of sequential Monte Carlo inference for probabilistic programming languages. In *ESOP*, pages 404–431, 2021.

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.

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.

Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv preprint arXiv:1404.0099*, 2014.

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.

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.

George Matheos, Alexander K Lew, Matin Ghavamizadeh, Stuart Russell, Marco Cusumano-Towner, and Vikash Mansinghka. Transforming worlds: Automated involutive MCMC for open-universe probabilistic models. In *Third Symposium on Advances in Approximate Bayesian Inference*, 2020.

Medicare. Hospital Compare, 2012.

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.

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.

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

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

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

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.

Christian Naesseth, Scott Linderman, Rajesh Ranganath, and David Blei. Variational sequential Monte Carlo. In *International Conference on Artificial Intelligence and Statistics*, pages 968–977. PMLR, 2018.

Kirill Neklyudov, Max Welling, Evgenii Egorov, and Dmitry Vetrov. Involutive MCMC: a unifying framework. In *International Conference on Machine Learning*, pages 7273–7282. PMLR, 2020.

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

Brooks Paige, Frank Wood, Arnaud Doucet, and Yee Whye Teh. Asynchronous anytime sequential Monte Carlo. *Advances in Neural Information Processing Systems*, 27, 2014.

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

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

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.

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 Artificial Intelligence*, pages 1056–1066. PMLR, 2021.

Frank Wood, Jan Willem Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, pages 1024–1032. PMLR, 2014.

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

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

Appendix

This appendix contains:

- A measure-theoretic presentation of the key definitions and theorems (Appendix A).
- Omitted proofs (of Theorem 1, Theorem 2, and Proposition 2), in Appendices B, C, and D.
- Further details on automation (Appendix E), including:
  - A brief introduction to the standard techniques Gen uses to automate the `SAMPLE-TRACE`, `EVAL-LOGPDF`, and `COMPUTE-RETVAL` operations (Appendix E.1).
  - A description of our automated fuzz-testing algorithm for detecting user proposal programs that violate the conditions of Definition 3 (Appendix E.2).
  - An explanation of how Gen efficiently implements automated SMCP^3 updates (Algorithm 2) by exploiting cancellations in density ratios where possible (Appendix E.3)
- Full details on experimental setup, for each experiment in Section 5 (Appendix F). Code for reproducing the experiments is also available, at `https://github.com/probcomp/aistats2023-smcp3`.
- Additional experiments which compare inference algorithms at equal runtimes (Appendix G). (This is provided because the results in Section 5 compare algorithms at equal particle counts rather than equal runtimes.)

A Measure-Theoretic Presentation

In this Appendix, we give more rigorous statements of the key definitions and theorems.

Mathematical setting for SMC. Consider a sequence $\tilde{P}_1, \ldots, \tilde{P}_T$ of unnormalized target measures, defined over measurable spaces $X_1, \ldots X_T$. For each t, let μ_t be a reference measure with respect to which $\tilde{P}_t$ is absolutely continuous. We write $\tilde{p}_t$ for the unnormalized density $\frac{d\tilde{P}_t}{d\mu_t}$.

Definition 4 (Probabilistic computation)**.** A *probabilistic computation* $K : X \rightsquigarrow Y$ between measurable spaces X and Y is a tuple (U_K, μ_K, Q_K, f_K), where:

- U_K is a measurable space of auxiliary randomness.
- μ_K is a reference measure on U_K.
- $Q_K : X \to U_K$ is a probability kernel, such that $Q_K(x, du_K) \ll \mu_K(du_K)$ for all $x \in X$.
- $f_K : X \times U_K \to Y$ is a measurable map.

We write $q_K(x \to u_K)$ for the density of the kernel $Q_K(x, du_K)$ with respect to μ_K.

Definition 5 (SMCP^3 move)**.** An SMCP^3 move from $\tilde{P}_{t-1}$ to $\tilde{P}_t$ is a pair of probabilistic computations, $K_t : X_{t-1} \rightsquigarrow X_t \times U_L$ and $L_t : X_t \rightsquigarrow X_{t-1} \times U_K$, satisfying:

- **Full support**: The measure $\tilde{P}_t(dx_t)Q_{L_t}(x_t, du_{L_t})$ must be absolutely continuous with respect to $\int_{X_{t-1}} \int_{U_{K_t}} \tilde{P}_{t-1}(dx_{t-1})Q_{K_t}(x_{t-1}, du_{K_t})\delta_{f_{K_t}(x_{t-1}, u_{K_t})}(dx_t, du_{L_t})$.
- **Invertibility**: $\tilde{P}_{t-1}(dx_{t-1})Q_{K_t}(x_{t-1}, du_{K_t})$-almost-everywhere, $f_{L_t} \circ f_{K_t}$ should equal the identity map.

Definition 6 (Restriction of a measure)**.** Let μ, ν be measures on X. Then $R(\mu, \nu)$, the *restriction of μ to the support of ν*, is the measure mapping a set $E \in \mathcal{X}$ to $\inf_{A \in \mathcal{X}}\{\mu(E \setminus A) \mid \nu(A) = 0\}$.

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

In the definition below, we use the notation $\mu \otimes \nu$ to denote the product of two measures, and $\mu \otimes \kappa$ to denote the product of a measure with a kernel (that is, $\mu(dx)\kappa(x, dy)$). Furthermore, we use the standard notation $\mu \circ f^{-1}$ to mean the pushforward of μ by measurable map f.

Definition 7 (Reference measures ξ_K^t and ξ_L^t)**.** Given an SMCP^3 move (K, L) from $\tilde{P}_{t-1}$ to $\tilde{P}_t$, we define the reference measures

- ξ_K^t to be $R(\mu_{t-1} \otimes \mu_K, \tilde{P}_{t-1} \otimes Q_K)$, the restriction of the product reference measure on $X_{t-1} \times U_K$ to the support of $\tilde{P}_{t-1}(dx_{t-1})Q_K(x_{t-1}, du_K)$.
- ξ_L^t to be $R(\mu_t \otimes \mu_L, (\tilde{P}_{t-1} \otimes Q_K) \circ f_K^{-1})$, the restriction of the product reference measure on $X_t \times U_L$ to the support of the K proposal.

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

$$\hat{w} = \frac{\tilde{p}_t(x')q_L(x' \to u_L)}{\tilde{p}_{t-1}(x)q_K(x \to u_K)} \cdot \frac{d\xi_L^t}{d(\xi_K^t \circ f_K^{-1})}(x', u_L).$$

We can also give a more precise statement of the local optimality result:

Proposition 1. Let $\nu = (\tilde{P}_{t-1} \otimes Q_{K_t}) \circ {f_{K_t}}^{-1}$. Then if Q_{L_t} is such that $\nu = (\nu \circ \pi_1^{-1}) \otimes Q_{L_t}$, then Q_{L_t} is locally optimal, in that it minimizes the variance of the incremental weight $\hat{w}$.

In our development from Section 4, we take $U_K = \mathbb{T}$, the space of program traces. We now define this object more precisely.

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 $(V_\tau, \mathcal{V}_\tau, \mu_\tau)$ of values, where μ_τ is a reference measure. For the reals, we choose $\mu_\mathbb{R} = \Lambda$ (the Lebesgue measure), and for discrete types we choose the counting measure. We let $\mathcal{K}$ 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 8. A *trace shape* $s \in \mathbb{S}$ is a finite set of entries (k, τ), such that $k \in \mathcal{K}$ 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 9. A *trace* $t \in \mathbb{T}$ is a trace shape s and a tuple $\mathbf{v} \in \bigtimes_{(k,\tau)\in s} V_\tau$, with one value for each name in s.

We equip $\mathbb{T}$ 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 $\bigtimes_{(k,\tau)\in s} V_\tau$. For each trace shape s, we can define the product reference measure $\mu_s = \bigotimes_{(k,\tau)\in s} \mu_\tau$. Then define reference measure $\mu_\mathbb{T}$ over all traces as

$$\mu_\mathbb{T}(B) = \sum_{s\in\mathcal{S}} \mu_s(\{\mathbf{v} \mid (s, \mathbf{v}) \in B\}).$$

For any Gen program P, the kernel $Q_P : A \rightsquigarrow \mathbb{T}$ has a density with respect to $\mu_\mathbb{T}$.

The formal version of Theorem 2 from the main paper is stated and proved in Appx. D.

B Proof of Theorem 1

Lemma 1. Let P and Q be probability measures on spaces X and Y respectively. Let $\tilde{Q} = Z_Q \cdot Q$ and $\tilde{P} = Z_P \cdot P$ denote unnormalized versions of these measures, with densities $\tilde{p}$ and $\tilde{q}$ with respect to reference measures μ_X and μ_Y. Let $S = Q \circ f^{-1}$ for some measurable bijection $f : X \to Y$, 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 $y = f(x)$, $x \sim Q$, and $w = \frac{\tilde{p}(y)}{\tilde{q}(x)} \cdot \frac{dR(\mu_Y, S)}{d(R(\mu_X, Q)\circ f^{-1})}(y)$.

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

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

On the first line, we use the fact that since P is absolutely continuous with respect to both μ_Y and S, it is absolutely continuous with respect to $R(\mu_Y, S)$. On the second line, we use the fact that since $S \ll \mu_X \circ f^{-1}$ and $S \ll Q \circ f^{-1}$, $S \ll (R(\mu_X, Q) \circ f^{-1})$, and therefore $R(\mu_Y, S) \ll S \ll (R(\mu_X, Q) \circ f^{-1})$.

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 $g : Y \to \mathbb{R}_{\geq 0}$, we have $\mathbb{E}[wg(y)] = \int \frac{Z_P}{Z_Q}\frac{dP}{dS}(y)g(y)S(dy) = \int g(y)(\frac{Z_P}{Z_Q}P)(dy)$. 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 $u_K \sim Q_K(x \to \cdot)$, and $(x', u_L) = f_K(x, u_K)$, the pair $(x', \hat{w} \cdot w)$ is properly weighted for $\tilde{P}_t$, where

$$\hat{w} = \frac{\tilde{p}_t(x')q_L(x' \to u_L)}{\tilde{p}_{t-1}(x)q_K(x \to u_K)} \cdot \frac{dR(\mu_t \otimes \mu_L, (\tilde{P}_{t-1} \otimes Q_K) \circ f_K^{-1})}{d(R(\mu_{t-1} \otimes \mu_K, \tilde{P}_{t-1} \otimes Q_K) \circ f_K^{-1})}(x', u_L).$$

Proof. We apply Lemma 1, with $X := X_{t-1} \times U_K$, $Y := X_t \times U_{L_t}$, $f := f_K$, $\tilde{P} := \tilde{P}_t \otimes Q_L$, and $\tilde{Q} := \tilde{P}_{t-1} \otimes Q_K$. This tells us that for all measurable $g : X_t \times U_L \to \mathbb{R}_{\geq 0}$, $\mathbb{E}_{x \sim P_{t-1}, u_K \sim Q_K}[\hat{w} \cdot g(f_K(x, u_K))] = \int \frac{Z_{P_t}}{Z_{P_{t-1}}} g(x', u_L)(P_t \otimes Q_L)(d(x', u_L))$. In particular, for all $h : X_t \to \mathbb{R}_{\geq 0}$, we can take $g(x', u_L) = h(x')$, and get that $\mathbb{E}_{x \sim P_{t-1}, u_K \sim Q_K}[\hat{w} \cdot h(\pi_1(f_K(x, u_K)))] = \int \frac{Z_{P_t}}{Z_{P_{t-1}}} h(x')P_t(dx')$.

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):

$$\int \frac{Z_{P_t}}{Z_{P_{t-1}}} h(x')P_t(dx') = \mathbb{E}_{x \sim P_{t-1}, u_K \sim Q_K}[\hat{w} \cdot h(\pi_1(f_K(x, u_K)))] = \mathbb{E}_{(x,w)}[\frac{1}{Z_{P_{t-1}}} \cdot w \cdot \mathbb{E}_{u_K \sim Q_K}[\hat{w} \cdot h(\pi_1(f_K(x, u_K)))]].$$

By linearity of expectation, we can rewrite to get:

$$\frac{1}{Z_{P_{t-1}}} \int Z_{P_t} h(x')P_t(dx') = \frac{1}{Z_{P_{t-1}}} \mathbb{E}[w \cdot \hat{w} \cdot h(\pi_1(f_K(x, u_K)))].$$

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 $\mathbb{E}[w \cdot \hat{w} \cdot h(x')] = \int h(x')\tilde{P}_t(dx')$. □

C Proof of SMCP^3 Convergence

C.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 Chopin et al. (2020), but using slightly different notation and terminology for consistency with this paper.

Definition 10 (Feynman-Kac model.)**.** A Feynman-Kac model is a 4-tuple $(\mathbb{Q}_1, (Q_t)_{t=2}^T, G_1, (G_{t=t}^T))$, where

1. $\mathbb{Q}_1$ is a probability measure on a measurable space $\mathcal{X}_1$
2. $Q_t : \mathcal{X}_{t-1} \to \mathcal{X}_t$ is a kernel between 2 measurable spaces
3. $G_1 : \mathcal{X}_1 \to \mathbb{R}_{\geq 0}$ is a measurable "potential" function on $\mathcal{X}_1$
4. $G_t : \mathcal{X}_{t-1} \times \mathcal{X}_t \to \mathbb{R}_{\geq_0}$ is a measurable "potential" function on measurable product space $\mathcal{X}_{t-1} \times \mathcal{X}_t$

This model is said to *target* the sequence of measures $(\mathbb{P}_t)_{t=1}^T$ on $(\mathcal{X}_t)_{t=1}^T$, where for any measurable $A \subseteq \mathcal{X}_1$,

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

and for any $t > 1$, for any measurable $A \subseteq \mathcal{X}_t$,

$$\mathbb{P}_t(A) = \int_{\mathcal{X}_{t-1}} \int_A G_t(x_{t-1}, x_t) Q_t(x_{t-1} \to dx_t)\mathbb{P}_{t-1}(dx_{t-1}). \tag{2}$$

Note that in the definition of a Feynman-Kac model from Chopin et al. (2020), each space $\mathcal{X}_t$ must be the same space $\mathcal{X}$. This restriction is immaterial because given any Feynman-Kac model defined as above, we can obtain a Feynman-Kac model under the definition from Chopin et al. (2020) by taking $\mathcal{X} = \oplus_{t=1}^T \mathcal{X}_t$, and extending $\mathbb{Q}_1$ and G_1 and each Q_t and G_t to measures/kernels/functions on this full space. Note also that what we call $\mathbb{Q}_1$ and Q_t, Chopin et al. (2020) calls $\mathbb{M}_0$ and M_{t-1}, and what we call $\mathbb{P}_t$, Chopin et al. (2020) calls $\mathbb{Q}_{t-1}$.

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 $X_1, \ldots, X_T$ admitting densities $\tilde{p}_t$ w.r.t base measures μ_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. μ_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. 3), 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 μ_{K_t} and μ_{L_t}.

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

1. $\mathbb{Q}_1 = Q_1$
2. $Q_2(x_1 \to A) = \bar{Q}_2(x_1 \to A)$ and $\forall t > 2, Q_t((x_{t-1}, u_{t-1}) \to A) = \bar{Q}_t(x_{t-1} \to A)$
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, $\bar{Q}_t(x_{t-1} \to \cdot)$ is the kernel from $X_{t-1} \to X_t \times U_{L_t}$ implemented by sampling $u_{K_t} \sim Q_{K_t}(x_{t-1} \to \cdot)$ then running f_{K_t},

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

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

$$g_t(x, (x', u_L)) = \frac{\tilde{p}_t(x')q_{L_t}(x' \to u_L)}{\tilde{p}_{t-1}(x)q_{K_t}(x \to u_K)} \cdot \frac{d(\mu_t \otimes \mu_{L_t})}{d((\mu_{t-1} \otimes \mu_{K_t}) \circ f_{K_t}^{-1})}(x', u_L) \qquad \text{where } (\_, u_K) = f_{L_t}(x', u_L).$$

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 $A \subseteq X_t \times U_{L_t}$. The target measure on this set is

$$\mathbb{P}_t(A) = (\tilde{P}_t \otimes Q_{L_t})(A) := \int_{X_t} \int_{U_{L_t}} 1_{(x_t, u_{L_t}) \in A} Q_{L_t}(x_t \to du_{L_t})\tilde{P}_t(dx_t)$$

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

$$
\begin{aligned}
\bar{\mathbb{P}}_t(A) &= \int_{X_{t-1}\times U_{L_{t-1}}} \int_A g_t(x_{t-1},(x_t,u_{L_t}))\bar{Q}_t(x_{t-1}\to dx_t, du_{L_t})\mathbb{P}_{t-1}(dx_{t-1}, du_{L_{t-1}}) \\
&= \int_{X_{t-1}} \int_{U_{L_{t-1}}} \int_A g_t(x_{t-1},(x_t,u_{L_t}))\bar{Q}_t(x_{t-1}\to dx_t, du_{L_t})Q_{L_{t-1}}(x_{t-1}\to du_{L_{t-1}})\tilde{P}_{t-1}(dx_{t-1}) \\
&= \int_{X_{t-1}} \int_A g_t(x_{t-1},(x_t,u_{L_t}))\bar{Q}_t(x_{t-1}\to dx_t, du_{L_t})\tilde{P}_{t-1}(dx_{t-1}) \\
&= \int_{X_{t-1}} \int_{U_{K_t}} 1_{f_{K_t}(x_{t-1},u_{K_t})\in A} g_t(x_{t-1}, f_{K_t}(x_{t-1},u_{K_t}))\tilde{p}_{t-1}(x_{t-1})Q_{K_t}(x_{t-1}\to du_{K_t})\mu_{t-1}(dx_{t-1}) \\
&= \int_{X_{t-1}} \int_{U_{K_t}} 1_{f_{K_t}(x_{t-1},u_{K_t})\in A} g_t(x_{t-1}, f_{K_t}(x_{t-1},u_{K_t}))q_{K_t}(x_{t-1}\to u_{K_t})\tilde{p}_{t-1}(x_{t-1})\mu_{K_t}(du_{K_t})\mu_{t-1}(dx_{t-1}) \\
&= \int_{f_{K_t}^{-1}(A)} \frac{d(\mu_t\otimes\mu_{L_t})}{d((\mu_{t-1}\otimes\mu_{K_t})\circ f_{K_t}^{-1})}(f_{K_t}(x_{t-1},u_{K_t}))(\mu_{t-1}\otimes\mu_{K_t})(dx_{t-1}, du_{K_t}) \\
&\qquad \cdot q_{K_t}(x_{t-1}\to u_{K_t})\tilde{p}_{t-1}(x_{t-1})\cdot\frac{\tilde{p}_t(x_t)q_{L_t}(x_t\to u_{L_t})}{q_{K_t}(x_{t-1}\to u_{K_t})\tilde{p}_{t-1}(x_{t-1})} \quad \text{where } (x_t,u_{L_t}) = f_{K_t}(x_{t-1},u_{K_t}) \\
&= \int_A \frac{d(\mu_t\otimes\mu_{L_t})}{d((\mu_{t-1}\otimes\mu_{K_t})\circ f_{K_t}^{-1})}(x_t,u_{L_t})((\mu_{t-1}\otimes\mu_{K_t})\circ f_{K_t}^{-1})(dx_t, du_{L_t})\cdot\tilde{p}_t(x_t)q_{L_t}(x_t\to u_{L_t}) \\
&= \int_A \tilde{p}_t(x_t)q_{L_t}(x_t\to u_{L_t})d(\mu_t\otimes\mu_{L_t})(dx_t, du_{L_t}) \\
&= \int_{X_t}\int_{U_{L_t}} 1_{(x_t,u_{L_t})\in A} Q_{L_t}(x_t\to du_{L_t})\tilde{P}_t(dx_t) \\
&= \mathbb{P}_t(A)
\end{aligned}
$$

□

C.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 the incremental weights $\hat{w}_i^t$ in Alg. 1 are bounded above, then for any continuous, bounded function $\varphi : X_t \to \mathbb{R}$, there exists σ s.t.

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

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 Chopin et al. (2020), 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 $\sigma > 0$,

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

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

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

□

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 the definition of PAP functions on Euclidean spaces from Lee et al. (2020), then use the tools from Huot et al. (2023) to extend the definition to spaces of traces.

Definition 11. 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 12. An *analytic set* $A \subseteq \mathbb{R}^n$ is a set of the form $\{x \in U \mid f_1(x) \geq 0 \wedge \cdots \wedge f_k(x) \geq 0\}$, 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 13. 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 13 applies to partial functions, so this is not an issue.) Using Definition 13, 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 14. 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 $f : \sqcup_{j \in J} X_j \rightharpoonup \sqcup_{k \in K} X_k$ 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 $\mathbb{T} = \sqcup_{s \in \mathbb{S}}(\bigtimes_{(k,\tau) \in s} V_\tau)$. Since each $V_\tau \subseteq \mathbb{R}$, we can view $\mathbb{T}$ as a particular subset of $\sqcup_{s \in \mathbb{S}} \mathbb{R}^{|s|}$, and similarly $\mathbb{T} \times \mathbb{T}$ as a subset of $\sqcup_{(s_1,s_2) \in \mathbb{S} \times \mathbb{S}} \mathbb{R}^{|s_1|+|s_2|}$. 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 $\hat{f} : \sqcup_{(s_1,s_2) \in \mathbb{S} \times \mathbb{S}} \mathbb{R}^{|s_1|+|s_2|} \rightharpoonup \sqcup_{(s_1,s_2) \in \mathbb{S} \times \mathbb{S}} \mathbb{R}^{|s_1|+|s_2|}$, 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 $\mathbb{T} \times \mathbb{T}$.

Lemma 2. Let μ, ν be measures over X and Y, and $f : X \to Y$ a measurable bijection, such that $\nu \ll (\mu \circ f^{-1})$. Let $\{A_i\}_{i \in I}$ be a countable family of subsets of X such that $X \setminus \cup_{i \in I} A_i$ has μ-measure-zero. Further, let $g : Y \to \mathbb{R}_{\geq 0}$ 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 μ_i is the restriction of μ to A_i, ν_i is the restriction of ν 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 ν with respect to $\mu \circ f^{-1}$.

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 $\mu_\mathbb{T} \otimes \mu_\mathbb{T}$ to these subsets. Suppose $f : \mathbb{T}_1^2 \to \mathbb{T}_2^2$ is a PAP bijection and that μ_2 is absolutely continuous with respect to $\mu_1 \circ f^{-1}$. Then

$$\frac{d\mu_2}{d(\mu_1 \circ f^{-1})}(f(\tau_1, \tau_2)) = |\det(Jf_{\tau_1,\tau_2})(\rho(\tau_1) +\!\!+ \rho(\tau_2))|,$$

where $\rho : \mathbb{T} \to \sqcup_{d \in \mathbb{N}} \mathbb{R}^d$ extracts all the real-valued entries in a trace into a vector, $\phi_\tau : \mathbb{R}^{|\rho(\tau)|} \to \mathbb{T}$ replaces the real entries in τ 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 $\tau_1', \tau_2' = f(\phi_{\tau_1}(\mathbf{v}_1), \phi_{\tau_2}(\mathbf{v}_2))$, and then returns the vector concatenation $\rho(\tau_1') +\!\!+ \rho(\tau_2')$.

Proof. We will use Lemma 2 to carve $\mathbb{T}_1^2$ 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 $f_{\mathbf{t}_D^O}^{\mathbf{t}_D^I}$ is a PAP bijection from $U \subseteq \mathbb{R}^n \to V \subseteq \mathbb{R}^m$, for some n and some m. Then there is some partition of U into analytic subsets A_j of $\mathbb{R}^n$; let $J_{(\mathbf{t}_D^I, \mathbf{t}_D^O)}$ be the index set of this partition. Then let $\mathcal{I} = \{(\mathbf{t}_D^I, \mathbf{t}_D^O, j) \mid j \in J_{(\mathbf{t}_D^I, \mathbf{t}_D^O)} \wedge \Lambda(A_j^{(\mathbf{t}_D^O, \mathbf{t}_D^I)}) > 0\}$, and define $\mathbb{T}^2_{(\mathbf{t}_D^I, \mathbf{t}_D^O, j)}$ 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 $(\mathbf{t}_D^I, \mathbf{t}_D^O, j)$,

$$\frac{d\mu_2 \,|_{f(\mathbb{T}^2_{(\mathbf{t}_D^I, \mathbf{t}_D^O, j)})}}{d(\mu_1 \,|_{\mathbb{T}^2_{(\mathbf{t}_D^I, \mathbf{t}_D^O, j)}} \circ f^{-1}_{(\mathbf{t}_D^I, \mathbf{t}_D^O, j)})}(f(t_1, t_2)) = |\det(Jf_{t_1,t_2}(\rho(t_1) +\!\!+ \rho(t_2)))|.$$

First, note that restricted to $\mathbb{T}^2_{(\mathbf{t}_D^I, \mathbf{t}_D^O, j)}$, $g(t_1, t_2) := \rho(t_1) +\!\!+ \rho(t_2)$ 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:

$$\begin{aligned}
\frac{d\mu_2 \,|_{f(\mathbb{T}^2_{(\mathbf{t}_D^I, \mathbf{t}_D^O, j)})}}{d(\mu_1 \,|_{\mathbb{T}^2_{(\mathbf{t}_D^I, \mathbf{t}_D^O, j)}} \circ f^{-1}_{(\mathbf{t}_D^I, \mathbf{t}_D^O, j)})}(f(t_1, t_2)) &= \frac{d(\mu_2 \,|_{f(\mathbb{T}^2_{(\mathbf{t}_D^I, \mathbf{t}_D^O, j)})} \circ g^{-1})}{d(\mu_1 \,|_{\mathbb{T}^2_{(\mathbf{t}_D^I, \mathbf{t}_D^O, j)}} \circ f^{-1}_{(\mathbf{t}_D^I, \mathbf{t}_D^O, j)} \circ g^{-1})}(g(f(t_1, t_2))) \\
&= \frac{d(\mu_2 \,|_{f(\mathbb{T}^2_{(\mathbf{t}_D^I, \mathbf{t}_D^O, j)})} \circ g^{-1})}{d((\Lambda \circ (g^{-1})^{-1}) \circ f^{-1}_{(\mathbf{t}_D^I, \mathbf{t}_D^O, j)} \circ g^{-1})}(g(f(t_1, t_2))) \\
&= \frac{d\Lambda}{d(\Lambda \circ (f_{t_1,t_2} \,|_{g(\mathbb{T}^2_{(\mathbf{t}_D^I, \mathbf{t}_D^O, j)})})^{-1})}(f_{t_1,t_2}(g(t_1, t_2))),
\end{aligned}$$

where the first line uses the fact that g is a bijection, the second and third use the fact that $\mu_1 \circ (g^{-1})^{-1}$ is the Lebesgue measure on the image (under g) of the support of μ_1,[11] and the third also uses the fact that $f_{t_1,t_2} = g \circ f \circ g^{-1}$. Since f is PAP, $f_{t_1,t_2} \,|_{g(\mathbb{T}^2_{(\mathbf{t}_D^I, \mathbf{t}_D^O, j)})}$ 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.

□

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

E PPL Automation

E.1 Automation of Standard PPL Operations

Sampling and evaluating densities of traces. The Gen probabilistic programming system (Cusumano-Towner et al., 2019) 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 the sampling distribution over traces induced by program P, with respect to the reference measure $\mu_{\mathbb{T}}$. Roughly, these procedures work by overriding the behavior of $\{\texttt{name}\} \sim \texttt{distribution}$ statements:

- $\texttt{SAMPLE-TRACE}(P, a)$: Initialize storage for an empty trace τ, and run the program P with arguments a. At each $\{\texttt{name}\} \sim \texttt{distribution}$ statement, draw a sample from the distribution and record it (and its type) in the trace τ under $\texttt{name}$. When the program is finished, return the trace τ.
- $\texttt{EVALUATE-LOGPDF}(P, a, \tau)$: Given a program P with arguments a, and a trace τ sampled from the program with those arguments, initialize a weight to 1. At each $\{\texttt{name}\} \sim \texttt{distribution}$ statement, lookup the value v associated with the key $\texttt{name}$ in the trace τ, and multiply the weight by the density of $\texttt{distribution}$ at v. If any $\texttt{name}$ is missing, or if after finishing execution the trace contains unvisited $\texttt{name}$s, return 0; otherwise, return the final weight.
- $\texttt{COMPUTE-RETVAL}(P, a, \tau)$: Given a program P with arguments a, and a trace τ sampled from the program with those arguments, 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.

E.2 Fuzz-Testing the Conditions of SMCP^3 Moves

Users of SMCP^3 specify SMCP^3 moves as pairs of K and L probabilistic programs, subject to certain conditions (Definition 3). These conditions do not hold trivially of any proposal programs the user could write, so users may implement buggy proposals that fail to satisfy them. To aid users in catching this sort of bug, we present two automated tests, which fail with positive probability if and only if the conditions of Definition 3 are not satisfied. By running the tests repeatedly, users can become increasingly confident in the soundness of their proposals.

Testing absolute continuity ("Full Support" condition). To test the first criterion, absolute continuity, we repeatedly:

1. **Generate a random trace τ_P^t from the support of $\tilde{P}_t$.**

   Gen's $\texttt{GENERATE}$ method can be run on the model program P, with argument t and observation trace $\mathbf{y}_t$, to perform a constrained execution of P, sampling variables freely if they do not appear in $\mathbf{y}_t$, and otherwise constraining the variables to the values they take in $\mathbf{y}_t$.

   The effect is to produce a trace τ_P^t of $P(t \to \cdot)$; the guarantee that Gen makes is that the posterior $P^{\mathbf{y}_t}$ is absolutely continuous with respect to the distribution that $\texttt{GENERATE}$ induces. This means that we can perform rejection sampling with $\texttt{GENERATE}$ as the proposal until we find a τ_P^t that has positive unnormalized density $\tilde{p}_t(\tau_P^t)$. (In general this will only take one sample, but may take longer, e.g. if the addresses in $\mathbf{y}_t$ do not appear in every execution.)

2. **Run the L program to obtain corresponding traces τ_P^{t-1} and τ_K for $\tilde{P}_{t-1}$ and K.**

   We generate τ_L from $\texttt{SAMPLE-TRACE}(L, \tau_P^t)$, and then set (τ_P^{t-1}, τ_K) to $\texttt{COMPUTE-RETVAL}(L, \tau_P^t, \tau_L)$. This yields L's guess at *how* the trace τ_P^t could be proposed: what would the previous particle have been, and what would K have done to it?

3. **Check that τ_P^{t-1} and τ_K have positive density.**

   Run $\texttt{EVALUATE-LOGPDF}(P, t-1, \tau_P^{t-1})$ and $\texttt{EVALUATE-LOGPDF}(K, \tau_P^{t-1}, \tau_K)$, and fail if either call returns $-\infty$ (indicating that the density is 0).

Together, steps 1 and 2 have some chance of generating any pair of traces with positive density $\tilde{p}_t(\tau_P^t) q_L(\tau_P^t \to \tau_L)$. Then step 3 checks to see whether the "full support" condition is satisfied for those traces.

Testing that f_K and f_L are inverses almost everywhere. To test this criterion, we repeatedly generate inputs to f_K and check that they 'round-trip':

1. **Generate a random trace τ_P^{t-1} from the support of $\tilde{P}_{t-1}$.**

   We use the same strategy as in the previous test, rejection sampling with `GENERATE` as a proposal.

2. **Obtain a trace τ_K of the K program.**

   We generate τ_K from $\texttt{SAMPLE-TRACE}(K, \tau_P^{t-1})$.

3. **Check consistency of f_K and f_L.**

   Run $\texttt{COMPUTE-RETVAL}(K, \tau_P^{t-1}, \tau_K)$ to obtain (τ_P^t, τ_L), then run $\texttt{COMPUTE-RETVAL}(L, \tau_P^t, \tau_L)$ and check that it returns exactly (τ_P^{t-1}, τ_K). Fail if not.

E.3 Efficient Implementation of SMCP^3 Updates

The automated SMPC^3 update presented in Algorithm 2 has sub-optimal time complexity in cases where evaluating the density of the old and new model traces at steps $t-1$ and t repeats execution of the same code in the model program P. For example, this occurs when P represents a time series model such as the dynamics model in Figure 2(a), where the argument t denotes the number of timesteps that the model is unrolled for. In such cases, Algorithm 2 will require executing `EVALUATE-LOGPDF` on program P twice, first for $t-1$ timesteps, then for t timesteps, leading to a time complexity of $O(t)$ for a single SMCP^3 update, and $O(T^2)$ complexity for the full SMCP^3 algorithm (Algorithm 1).

To achieve asymptotic speed-ups over this naive implementation, Gen supports incremental computation that exploits the cancellation of density ratios to compute the desired incremental importance weights. This is provided by the `UPDATE-TRACE` procedure described below, which performs incremental computation for models written using Gen's modeling combinators or static modeling language (Cusumano-Towner et al., 2019). In addition, Gen provides a `SAMPLE-AND-EVALUATE-LOGPDF` procedure, which avoids the need to run `SAMPLE-TRACE` and `EVALUATE-LOGPDF` separately, leading to further (constant factor) speed-ups:

- $\texttt{UPDATE-TRACE}(P, a', \tau, \Delta\tau)$: Given a program P with *updated* arguments a', a trace τ sampled from P under the previous arguments a, and a *partial* trace $\Delta\tau$ that specifies the values of new or updated random variables, create a fresh copy of τ called τ' (the updated trace), and initialize an incremental weight w to 1. Identify (via static analysis, etc.) the set of computation paths ΔP that either: (i) would be executed if P were run with arguments a' instead of the previous arguments a; (ii) need to be executed to ensure all random variables in $\Delta\tau$ are sampled (it is an error if this is not possible). Execute *only* those computation paths.

  At each $\{\texttt{name}\} \sim \texttt{distribution}$ statement, check that `name` is present in either $\Delta\tau$ or τ. If not, this is an error. If present in $\Delta\tau$, store the corresponding value v' from $\Delta\tau$ in τ', replacing the previous value v copied from τ if any. Multiply the weight w by the density of `distribution` at v', and if a previous value v existed, divide w by the density of `distribution` at v. After execution of all incremental computation paths ΔP, return the updated trace τ', and the incremental weight w. The trace $\tau' = \tau \oplus \Delta\tau$ is equal to the return value of $\texttt{SAMPLE-TRACE}(P, a')$, and w is exactly equal to $\texttt{EVALUATE-LOGPDF}(P, a', \tau') - \texttt{EVALUATE-LOGPDF}(P, a, \tau)$.

- $\texttt{SAMPLE-AND-EVALUATE-LOGPDF}(P, a)$: Initialize storage for an empty trace, set weight w to 1, and run the program P with arguments a. At each $\{\texttt{name}\} \sim \texttt{distribution}$ statement, draw a sample v from the distribution and record it (and its type) in a trace under `name`. Multiply the weight w by the density of `distribution` at the sampled value v. When the program is finished, return the trace and the weight.

Using these two procedures, we can implement an efficient version of the SMCP^3 update, shown in Algorithm 4. Several key differences are as follows:

- The update takes in a trace of *additional* observations Δy_t, where $y_t = y_{t-1} \oplus \Delta y_t$ (i.e. observations in Δy_t are either added to y_{t-1} or replace previous observations).[12]
- Instead of separately sampling and evaluating the density of a trace u_{K_t} of the forward program K_t, we do this simultaneously using `SAMPLE-AND-EVALUATE-LOGPDF`.
- The first return value of K_t (and L_t) is allowed to be a *partial* trace Δx, which specifies the random variables that should be added to or replaced in the original trace x s.t. the new trace $x' = x \oplus \Delta x \oplus \Delta y_t$.

[12] Generalizations to both Algorithm 4 and `UPDATE-TRACE` are possible for the case where observations are *deleted* rather than added, but care must be taken to ensure support matching, and other subtleties apply.

Algorithm 4 Efficient implementation of automated SMCP^3 update

Require: model Gen program P
Require: additional observations $\Delta y_t = y_t \ominus y_{t-1}$
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 $\tilde{P}^{y_t}(t \to \cdot)$

$(u_{K_t}, \log q_{K_t}) \leftarrow \texttt{SAMPLE-AND-EVALUATE-LOGPDF}(K_t, x)$
$((\Delta x, u_{L_t}), \hat{J}) \leftarrow \texttt{AD}(\texttt{COMPUTE-RETVAL}(K_t, x, u_{K_t}))$
$(x', \log \frac{\tilde{p}_t}{\tilde{p}_{t-1}}) \leftarrow \texttt{UPDATE-TRACE}(P, t, x, \Delta x \oplus y_t)$
$\log q_{L_t} \leftarrow \texttt{EVALUATE-LOGPDF}(L_t, x', u_{L_t})$
$\log r \leftarrow \log \frac{\tilde{p}_t}{\tilde{p}_{t-1}} + \log q_{L_t} - \log q_{K_t}$
$\log w' \leftarrow \log w + \log r + \log |\det \hat{J}|$
return $(x', \log w')$

- Rather than running `EVALUATE-LOGPDF` on x and x' separately, we compute the incremental weight $\log \frac{\tilde{p}_t}{\tilde{p}_{t-1}}$ and new trace x' using `UPDATE-TRACE`.

Due to `UPDATE-TRACE`, SMCP^3 using Algorithm 4 is asymptotically more efficient than using Algorithm 2 in many cases. For example, in the time series model from Figure 2(a) or the mixture model in Section 5.2, `UPDATE-TRACE` executes only the code required to evaluate the density of the newest timestep or observation, which takes $O(1)$ time. As a result, the full SMCP^3 algorithm (Algorithm 1) has $O(T)$ instead of $O(T^2)$ time complexity.

F Experimental Details

All experiments were run on a commodity laptop with 32GB of RAM, using 8 cores. Every experiment presented in Sec. 5 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 5.1.

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 $\mathbb{R}^d \times \mathbb{R}^d$ and the space U_{L_t} is $\mathbb{R}^d$. Algorithm 5 gives the K and L proposals for this algorithm in mathematical notation. Given particle $\mathbf{z}_{1:t-1}$, the K kernel first samples an auxiliary $\mathbf{v}$ from the dynamics prior, then samples $\mathbf{z}_t$ via a step of Langevin ascent starting from $\mathbf{v}$. It returns the updated particle $\mathbf{z}_{1:t}$, and the auxiliary variable $\mathbf{v}$. The L kernel generates $\mathbf{v} \sim P(\cdot; z_{t-1})$ (ignoring $\mathbf{z}_t$).

Algorithm 5 SMCP^3 kernels for Langevin particle extension.

Require: Langevin ascent step-size σ.
Require: Observation trajectory $\mathbf{y}_{1:t}$.
procedure K($\mathbf{z}_{1:t-1}$)
  $\mathbf{v} \sim P_{dyn}(\cdot \mid \mathbf{z}_{t-1})$ ▷ Sample an auxiliary position from the dynamics prior
  ▷ Sample $\mathbf{z}_t$ via a step of Langevin ascent, starting from $\mathbf{v}$.
  $\mathbf{z}_t \sim \mathcal{N}(\mathbf{v} + \sigma^2 \frac{\partial \log P(Z_t = \mathbf{v}, \mathbf{y}_t | \mathbf{z}_{t-1})}{\partial \mathbf{v}}), \sqrt{2}\sigma I)$
  return $(\mathbf{z}_{1:t}, \mathbf{v})$
end procedure
procedure L($\mathbf{z}_{1:t}$)
  $\mathbf{v} \sim P(Z_t = \cdot | \mathbf{z}_{t-1})$
  return $(\mathbf{z}_{1:t-1}, (\mathbf{v}, \mathbf{z}_t))$
end procedure

Experiment details. Fig. 3 shows an artificial 1-D observation trajectory generated by adding $\mathcal{N}(0, 0.8)$ noise to each position in the hand-written vector $[[0, 3, 6, 9, 12, 9, 12, 16, 18, 18, 20, 17]$. For this trajectory, Fig. 3 shows the 3-σ band of the exact filtering posterior distribution over latent states at each timestep (obtained via Kalman filtering), and the values of a single particle over time from our SMCP^3 algorithm. The bottom two panels compare SMCP^3 against two baselines: a bootstrap particle filter, and a bootstrap filter with MALA rejuvenation on $\mathbf{z}_t$ at each step t. All algorithms were run with $N = 5$ particles. For the SMCP^3 algorithm and MALA, we use Langevin-ascent step size $\sigma = 0.3$. Resampling happens when ESS drops below $N/2$.

Fig. 5 shows 200 particles generated at $t = 1$ by each inference algorithm, in 1-D and given $y_1 = 5$. Each particle is rendered at its z_1 position on the x axis, and on the y axis, at a height so that it lands on the posterior density curve for $P(z_1|y_1)$.

Table 1 reports results of each algorithm on a 100-dimensional dataset, generated from the model, with 50 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 z_t is a latent state that evolves according to Gaussian random walk dynamics, $z_1 = 0$; $\forall t > 1, z_t \sim \mathcal{N}(z_{t-1}, 0.7)$. The observed data y_t is a "point-cloud" of 200 3D positions detected by a depth-camera (so $y_t \in \mathbb{R}^{3\times 200}$). The likelihood $P(y_t|z_t)$ 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 $q_t = [q_t^1, \ldots, q_t^M] \in \mathbb{R}^{3\times M}$. Then, the observed point-cloud y_t is generated from the following Gaussian mixture model:

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

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

SMCP^3 inference algorithm. Alg. 6 gives the K and L proposals used by our SMCP^3 algorithm. The K kernel first proposes a position v from the dynamics model, then chooses one of a finite number of points on a grid centered at v as the value for z_t; the L kernel fills in the value of v by enumerating each one which could have led to K returning z_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 SMCP^3 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. 5.2. We then detail the SMCP^3 inference algorithm.

F.2.1 Model, Baselines, and Experiment Details

Section 5.2 describes the DPMM we use for online clustering. It has two parameters: (1) the parameter α_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 measure space $(V, \mathcal{V}, \mu_V)$ of values, and $F : \oplus_{n\in\mathbb{N}} V^n \to \mathbb{R}_{\geq 0}$ is the function s.t. $F(\vec{v})$ is the joint probability density of vector $\vec{v}$ w.r.t. $\mu_V^{|\vec{v}|}$. (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 $\oplus_{n\in\mathbb{N}} V^n$.)

Algorithm 6 SMCP^3 kernels for enumeration-based particle extension.

Require: Grid step size ϵ and range N.
Require: Observation trajectory $y_{1:t}$.
procedure K($z_{1:t-1}$)
  $v \sim P(\cdot; z_{t-1})$ ▷ Sample an auxiliary position from the dynamics prior
  ▷ Enumerate over the grid, starting from v, and evaluate the probability of each state.
  $D \leftarrow \{\}$ ▷ Initialize score dictionary.
  for $i = -N, \ldots, N$ **do**
    $z_t^i \leftarrow v + i\epsilon$ ▷ Potential z_t value in grid.
    $s \leftarrow P(z_t^i | z_{t-1}) P(y_t | z_t^i)$ ▷ Joint probability score for this z_t value.
    $D \leftarrow D \cup \{z_t^i \mapsto s\}$
  end for
  Sample $z_t \sim P_D$, where P_D is the distribution with $P_D(x_t) \propto D[x_t]$.
  return $(z_{1:t}, v)$
end procedure
procedure L($z_{1:t}$)
  $D \leftarrow \{\}$ ▷ Initialize score dictionary.
  for $i = -N, \ldots, N$ **do**
    $v^i \leftarrow z_t + i\epsilon$ ▷ Potential v value in grid.
    $s \leftarrow P(v^i | z_{t-1})$
    $D \leftarrow D \cup \{v^i \mapsto s\}$
  end for
  Sample $v \sim P_D$, where P_D is the distribution with $P_D(u) \propto D[u]$.
  return $(z_{1:t-1}, (v, z_t))$
end procedure

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 $P(\Pi_t | y_{1:t})$, the posterior over possible partitions of data-indices $\{1, \ldots, t\}$, given the first t datapoints.

Mixture-model likelihoods. We use three different F likelihoods in our experiments, two for modeling real data (so $V = \mathbb{R}$, 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 $\mu \sim \mathcal{N}(\mu_0, \sigma_0^2)$ and a known variance σ^2. For any vector $\vec{y}$,

$$F_1(\vec{y}) = \int_{\mathbb{R}} \prod_{y \in \vec{y}} \mathcal{N}(y; \mu, \sigma^2) \mathcal{N}(d\mu; \mu_0, \sigma_0^2).$$

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

$$F_2(\vec{y}) = \int_{\mathbb{R}} \int_{\mathbb{R}} \sum_{y \in \vec{y}} \mathcal{N}(\vec{y}; \mu, \sigma^2) \Gamma(d(\sigma^{-2}); \alpha_y, \beta) \mathcal{N}(d\mu; \xi, \kappa^{-2}).$$

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 μ and σ parameters above, the string s^* is marginalized out in the likelihood function.) We use the exact likelihood defined in Lew et al. (2022) 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 $\{1, \ldots, t-1\}$) to a partition Π_t of $\{1, \ldots, t\}$, given datapoint y_t. The proposal distribution either creates a new cluster containing only t, outputting $\Pi_t = \Pi_{t-1} \cup \{\{t\}\}$, or adds t to an existing $C_* \in \Pi_{t-1}$ outputting $\Pi_t = \Pi_{t-1} \setminus \{C_*\} \cup \{C_* \cup \{t\}\}$. This proposal Q chooses among these $|\Pi_{t-1}| + 1$ options such that

$Q(\Pi) \propto p_t(\Pi, y_{1:t})$, 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 Miller et al. (2015), 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 involved, so we devote a subsection (Sec. F.2.2) to it below.

Experiment details. Table 1 shows the results of mixture-model inference in a data-cleaning dataset of 1k strings from Medicare records (Lew et al., 2021a).

To model the Medicare dataset, we used likelihood F_3, and set $\alpha_D = 1.0$. Table 1 shows the results of each SMC algorithm on this dataset, using 2 particles for each algorithm, with means and empirical variances taken over 2 SMC runs for the SMCP3 algorith, and 5 runs for each baseline.

Figure 3 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 $\alpha_D = 1.0$, using F_1 with $\mu_0 = 0$, $\sigma_0 = 6$, and $\sigma = 1$; we ran inference using $\alpha_D = 1.0$, $\mu_0 = 0$, $\sigma_0 = 10^6$, and $\sigma = 1$. 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 $C'_* = C_* \cup \{t\}$, 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 $C \in \Pi_t^1 \setminus \{C'_*\}$, 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 $C \in \Pi_t^1 \setminus \{C'_*\}$, and for each of these, computes $s_C = p_t(\Pi_t^C, y_{1:t})$, where $\Pi_t^C = \Pi_t^1 \setminus \{C'_*, C\} \cup \{C'_* \cup C\}$ 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

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

This is the sum over every possible way of splitting C'_* into two clusters C' and C'' of the joint PDF $p_t(\Pi_t^{C', C''}, y_{1:t})$ of the partition $\Pi_t^{C', C''}$ 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 $Q^K_{\text{split}}(C'_*, y_{1:t} \to u, C', C'')$ 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 $h^K(C'_*, C', C'', y_{1:t} \to u)$ 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 Lew et al. (2022); 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 $(C'_i, C''_i, u_i) \sim Q^K_{\text{split}}(C'_*, y_{1:t} \to \cdot)$, and computes the importance weight

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

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 $Q^K_{\text{split}}(C'_*, y_{1:t} \to u, C', C'')$ 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 Miller et al. (2015) 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 $p = -1$ deterministically, and finally initializes $C' = \{c_1\}$ and $C'' = \{c_2\}$.

Inverting the auxiliary randomness in a single split proposal. The kernel $h^K(C'_*, C', C'', y_{1:t} \to u)$ 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 $p = -1$, 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 $\Pi^1_{t-1} = \Pi_t \setminus \{\tilde{C}\} \cup \{\tilde{C}'\}$. 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 $\Pi^1_{t-1} \setminus \{\tilde{C}'\}$, 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 $p_t(\cdot, y_{1:t})$ is computed in L using $p_{t-1}(\cdot, y_{1:t-1})$, (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

	Est. of $\log P(\mathbf{y}_{1:T})$	Mean runtime	Particle count
Medicare data			
Locally Optimal SMC	-37294.94 $\pm$ 2650.37	42.2 seconds	160
Resample-Move Split/Merge	-14259.87 $\pm$ 274.92	42.3 seconds	32
SMCP^3 Split/Merge	**-13882.15 $\pm$ 0.53**	**34.0 seconds**	2

Table 2: Results for the model in section 5.2, with the particle counts and mean runtime for each SMC algorithm. Results are computed in the same way as those in table 1.

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 $u \sim h^L(C_* \cup C, C_*, C, y_{1:t-1} \to \cdot)$.

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 the split-related proposals in L.
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.)

G Runtime-normalized experimental results

In section 5, for simplicity, experimental results were presented in which the SMC algorithms under comparison were run with equal particle counts. In this section, we include algorithmic comparisons normalized by wall-clock runtime.

Online inference in state-space models. In Section 5.1, we compared SMC inference algorithms run with equal particle counts. Here we provide plots showing that similar quantitative results hold when we compare the inference algorithms each given similar runtimes. In particular, Figure 8 plots the performance of SMCP^3 and our two baseline algorithms on the 100-dimensional state-space modeling task from Section 5.1. Figure 7 also plots the performance of SMCP^3 and the baselines on the 3D object tracking task from Figure 6. For each of these examples, the runtime plots tell essentially the same story as the particle count plots.

Online inference in mixture models. Table 2 shows a runtime-normalized variant of the mixture model results from Table 1. As in the particle-count-normalized experiments from Table 1, on the Medicare dataset, SMCP^3 achieves better log-marginal-likelihood estimates than the baselines.

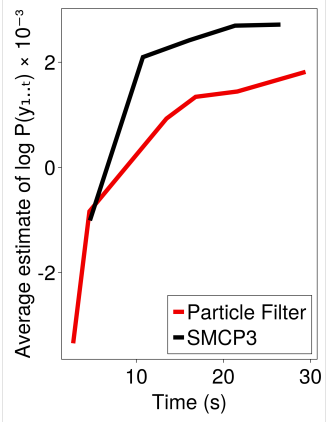

Figure 7: Log marginal likelihood vs. wall clock time for SMCP^3 and a bootstrap particle filter. The SMCP^3 algorithm was run with 2, 4, 6, 8, and 10 particles while the particle filter was run with 10, 18, 50, 63, 81, and 111 particles. To generate the 3D object-tracking results for this figure, inference was run on a generic CPU with no graphics optimizations, resulting in runtimes ranging between 10 to 30 seconds. Because the probabilistic model in the loop of the inference algorithm contains an approximate renderer, there is reason to expect that graphics optimizations and GPU rendering could significantly improve performance.

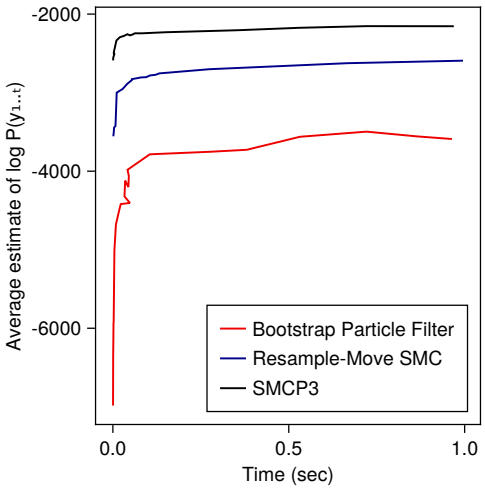

Figure 8: Results for the model in Section 5.1. We ran each algorithm with varying numbers of particles, between 1 and 2000, and measured average runtimes and average log marginal likelihood estimates. Here we plot log marginal likelihood estimates against runtimes.