Scalable Model-Based Clustering with Sequential Monte Carlo

Connie Trojan
Lancaster University

Pavel Myshkov
Microsoft Research

Paul Fearnhead
Lancaster University

James Hensman
Microsoft Research

Tom Minka
Microsoft Research

Christopher Nemeth
Lancaster University

Abstract

In online clustering problems, there is often a large amount of uncertainty over possible cluster assignments that cannot be resolved until more data are observed. This difficulty is compounded when clusters follow complex distributions, as is the case with text data. Sequential Monte Carlo (SMC) methods give a natural way of representing and updating this uncertainty over time, but have prohibitive memory requirements for large-scale problems. We propose a novel SMC algorithm that decomposes clustering problems into approximately independent subproblems, allowing a more compact representation of the algorithm state. Our approach is motivated by the knowledge base construction problem, and we show that our method is able to accurately and efficiently solve clustering problems in this setting and others where traditional SMC struggles.

1 INTRODUCTION

1.1 Knowledge Base Construction

There has been much recent interest in augmenting large language models with external knowledge. This has two key advantages over storing knowledge implicitly in the model's parameters: it avoids the need to fine-tune the model through additional training, allowing easy updating of knowledge over time; and it improves the explainablility of the model's output by giving explicit sources for its information. This external knowledge typically takes the form of a corpus of documents, however recent work by Wang et al. (2025) has focused on providing the information in the form of a structured knowledge base. This approach has a number of advantages over methods that work directly with a text corpus – it is more scalable than placing the entire corpus in the model's context window (corpus-in-context prompting, Lee et al., 2024), and does not require running an external retriever to extract relevant documents from the corpus (retrieval-augmented generation, Lewis et al., 2020).

However, it requires the additional step of constructing a suitable knowledge base from an unstructured text corpus. This process involves extracting structured entity fragments, consisting of partial observations of entities' attributes, and grouping them together to form entries in the knowledge base (Weikum and Theobald, 2010). The latter task is known as entity linking or entity disambiguation. When an appropriate reference database exists, this is typically seen as a classification task aiming to predict to which existing entity new observations should be attributed (Sevgili et al., 2022). However, many applications lack appropriate reference data, making the task of grouping entity fragments more challenging (Szekely et al., 2015; Winn et al., 2021). Since each fragment contains partial, free text observations of an entity's attributes, there can be ambiguity about which observations refer to the same entity (see Figure 1).

In this setting, knowledge base construction can be seen as a clustering problem: partial observations must be grouped into clusters, each containing observations about a different entity. Even finding a maximum likelihood clustering in this setting is difficult, since the number of possible clusterings increases exponentially with dataset size. An additional challenge is that inference must often be performed in the online setting, since new entities may appear and entity properties

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300. Copyright 2026 by the author(s).

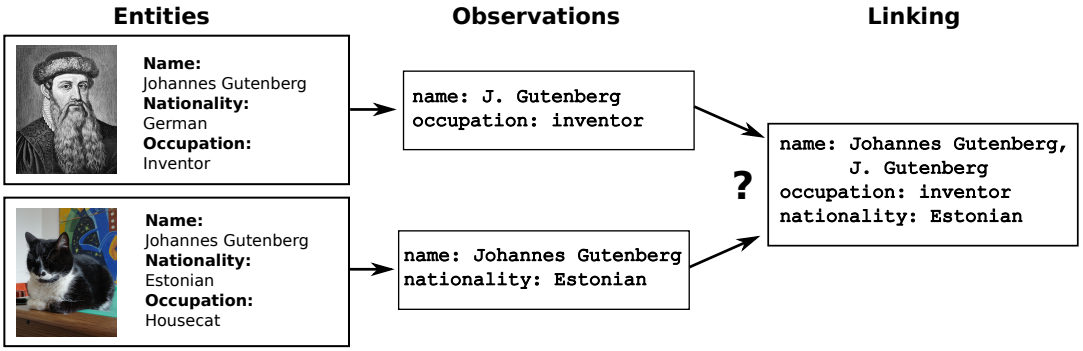

Figure 1: Entity disambiguation on Wikidata entries for "Johannes Gutenberg" (source: Kruusamägi, 2024; Wikimedia Foundation, 2012)

can change over time (Weikum and Theobald, 2010; Zaporojets et al., 2022). Existing algorithms are typically based on hierarchical clustering, merging fragments together in a greedy fashion based on pairwise comparisons (Benjelloun et al., 2009). However, this approach struggles in cases where there is significant uncertainty over possible clusterings that cannot be resolved until more information is observed, since merges cannot be undone without re-running the algorithm. As such, we focus on clustering algorithms designed for the online setting, which have the additional advantage of having reduced computational cost for large problems compared to offline methods.

1.2 Model-based clustering

Motivated by the knowledge base construction problem, we focus on solving clustering problems with a large, unknown underlying number of clusters, which require the use of a (potentially black-box) probabilistic model for cluster generation. We compare to both online and offline baselines in order to investigate the trade-off between accuracy, computational cost, and online capability.

Model-based clustering methods that do not require the number of clusters to be specified in advance are typically based on Bayesian nonparametric models, described in more detail in Section 2.1. Since the number of possible clusterings of the data (and hence the number of states in the posterior distribution) increases exponentially with dataset size, inference methods typically use sample-based approximations, or make simplifying assumptions in order to lower-bound the posterior density.

One commonly used approximate method is agglomerative hierarchical clustering (Jain and Dubes, 1988), which considers clusterings that can be obtained by starting with every observation in a singleton cluster and iteratively merging until some truncation/stopping condition is met. Potential merges can be evaluated by computing hypothesis test statistics (Banfield and Raftery, 1993; Heller and Ghahramani, 2005), with a natural stopping criterion given by the point at which no more proposed merges are accepted by the hypothesis test. This gives a greedy algorithm for estimating maximum likelihood or maximum a posteriori clusterings.

Approximate Bayesian inference in this setting spans a range of techniques. Variational inference methods (Kurihara et al., 2007; Scherreik, 2020) optimise a simplified posterior approximation, yielding fast, scalable updates that readily extend to streaming data, but at the expense of restrictive modelling assumptions and potentially reduced accuracy to the true posterior. Inference relies on assuming independence between all latent variables and making alternating updates to assignments and parameters, so cannot be directly applied on black-box likelihoods without explicitly defined cluster parameters.

By contrast, Monte Carlo algorithms sample cluster assignments exactly under the model, and do not impose restrictions on the form of the likelihood. Markov Chain Monte Carlo (MCMC) methods (Bouchard-Côté et al., 2017; Neal, 2000) iteratively sample individual assignments from their marginal distributions, but remain inherently offline and can converge slowly for large datasets. Distributed MCMC methods (Ge et al., 2015; Marchant et al., 2021) can accelerate convergence, but typically rely on problem features that may not hold in the black-box setting (e.g. availability of summary statistics). Sequential Monte Carlo (SMC) offers an online sampling alternative which incrementally samples a particle approximation of the posterior (Canini et al., 2009; Fearnhead, 2004). However, its memory and computational demands grow quickly with problem size (Kantas et al., 2015).

Our contribution We propose a novel online clustering framework that exploits the form of the posterior distribution to decompose large, complex datasets into independent subproblems and thereby enable efficient Sequential Monte Carlo inference. This problem decomposition is updated dynamically as new data arrive. Our method retains the asymptotic exactness of vanilla SMC, and we prove that the factorisation improves the reverse KL divergence to the full posterior, ensuring approximation fidelity is preserved. We demonstrate that our split SMC algorithm delivers substantial computational savings and improved clustering accuracy compared to standard approaches.

2 BACKGROUND

2.1 Bayesian Clustering Models

Bayesian nonparametric clustering methods allow the number of clusters k to grow with the data, making them well suited to tasks where the true cluster count is unknown. A canonical example is the Dirichlet process mixture model (DPMM), see Murphy (2023). Under a DPMM, we imagine data arriving sequentially: suppose that at time t, the existing data $x_{1:(t-1)}$ has cluster assignments $z_{1:(t-1)}$ partitioning it into clusters $\{c_1, c_2, \ldots, c_K\}$. A new observation x_t appears in an existing cluster with probability $\Pr(z_t = k \mid z_{1:(t-1)}) = \frac{|c_k|}{\alpha+t-1}$, or a fresh cluster is created with probability $\Pr(z_t = K+1 \mid z_{1:(t-1)}) = \frac{\alpha}{\alpha+t-1}$, where $|c_k|$ denotes the size of cluster k and $\alpha > 0$ is a concentration parameter controlling the rate at which new clusters appear. Once its cluster membership is decided, the data point x_t is drawn from the parametric distribution associated with its cluster, whose parameters themselves are drawn from a shared prior.

Inference under a DPMM typically involves evaluating, for a given data point x_i, the marginal probability of assignment to a given cluster conditional on the rest of the data and their clusters:

$$p(z_i = k \mid z_{-i}, x_{-i}) \propto \underbrace{\Pr(z_i = k \mid z_{-i})}_{\text{prior term}} \times \underbrace{p(x_i \mid x_{-i,k})}_{\text{likelihood term}}, \tag{1}$$

where z_{-i} and x_{-i} denote all assignments and observations except the ith, and $x_{-i,k}$ is the collection of these observations assigned to cluster k. The prior term follows directly from the above and the likelihood term requires integrating over any unknown cluster parameters. This likelihood can be obtained as the ratio $p(x_i, x_{-i,k})/p(x_{-i,k})$, where p refers to the likelihood of observing a given unordered collection of datapoints in a sample of the specified size. This marginal form underpins both Gibbs sampling (Neal, 2000) and particle-based sequential Monte Carlo (Section 2.2), since each update only requires local reassignment probabilities rather than enumeration of all partitions.

2.2 Sequential Monte Carlo

Sequential Monte Carlo (SMC) provides an online particle-based approximation to the evolving posterior distribution $p(z_{1:t} \mid x_{1:t})$, where the z_t are unobserved latent variables (in our case, cluster assignments) and x_t the corresponding observations (Doucet et al., 2001). In clustering problems, since the latent variables are discrete, it is possible to compute exact updates to the posterior distribution at each timestep. Fearnhead (2004) exploit this to formulate an SMC clustering algorithm that guarantees each particle represents a unique clustering, thus avoiding redundant duplicate particles.

In this algorithm, at each time t, we have access to a set of m particles $\{p_{t-1}^{(i)}\}_{i=1}^m$ with weights $\{w_{t-1}^{(i)}\}_{i=1}^m$ that approximate the posterior at time $t-1$. Each particle contains a distinct clustering of $x_{1:(t-1)}$, which can be represented as either a vector $z_{1:(t-1)}$ of cluster assignments, or a partition of $x_{1:(t-1)}$ into clusters. When a new data point x_t arrives, we perform three operations:

1. **Propagation.** Construct a set of *putative* particles, with one particle for each possible assignment of x_t to a cluster on each existing particle.
2. **Weighting.** For each putative particle $\tilde{p}_t(i, c_k)$, compute unnormalised putative weights using (1):
$$\tilde{w}_t(i, c_k) \propto w_{t-1}^{(i)}\, p(z_t = k \mid p_{t-1}^{(i)}, x_{1:(t-1)}).$$
3. **Resampling.** Draw m new particles $\{p_t^{(i)}\}$ and their weights $\{w_t^{(i)}\}$ by sampling from the putative particle set according to the putative weights.

This algorithm preserves the exact posterior up to the resampling step, which bounds both memory and computation by discarding low-probability configurations. Many resampling schemes are possible in this framework (Carpenter et al., 1999; Fearnhead and Clifford, 2003), trading off computational efficiency and variance reduction. Our algorithmic development (below) is based on making particle approximations that minimise reverse KL divergence to the true posterior. To be consistent with this, we use a simple greedy resampling scheme, where the m particles with the highest weights are selected and their weights normalised – this is the optimal way (in terms of reverse KL divergence) of resampling a particle approximation of fixed size m from a discrete distribution (Appendix B.2).

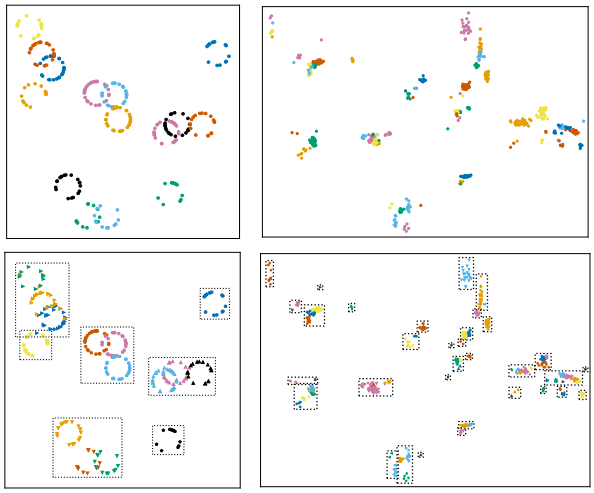

Figure 2: Top: scatterplots of the circles (left) and Gaussian mixture (right) datasets. Bottom: example split SMC clustering output, with subproblem partition indicated in dotted lines

3 SEQUENTIAL CLUSTERING METHODOLOGY

Overview Applying standard SMC directly to large problems quickly becomes infeasible: the number of plausible clusterings, and hence the number of particles required for accurate inference, grows exponentially with dataset size, significantly increasing computational and memory demands. Repeated resampling erodes uncertainty over earlier observations so that only the most recent data remain well-represented. Yet, under a Dirichlet process prior, well-separated subsets of the data are close to independent: if they have vanishing probability of sharing a cluster, their joint posterior factorises. By partitioning the dataset into these approximately independent subproblems and maintaining separate particle approximations for each, we can implicitly represent a much larger particle set, dramatically cutting both memory and computational costs. In addition, updating and resampling only the subproblem affected by each new observation allows our algorithm to preserve uncertainty elsewhere.

Our algorithm proceeds similarly to that of Fearnhead (2004) described in Section 2.2, with the addition of a splitting step (Section 3.2) at the end of each iteration to check whether the clustering problem can be decomposed. After such a split has occurred, we use a modified update step (Sections 3.3 and 3.4) that operates on the factorised representation.

3.1 Notation

We represent a clustering of the first t observations $x_{1:t}$ by a partition $p_t = \{c_1, c_2, \ldots, c_K\}$, where each cluster $c_j \subseteq_{\mathrm{m}} x_{1:t}$ and $\bigcup_{j=1}^{K} c_j = x_{1:t}$. We use the subscript m on set operations to clarify that we work with multisets of datapoints, since we consider datapoints with different time indices to be distinct from one another regardless of whether they have the same value. At time t, a particle approximation to the posterior has one such partition on each particle and consists of a weighted set

$$\mathcal{P}_t = \{\, p_t^{(i)} \}_{i=1}^{m}, \qquad \{\, w_t^{(i)} \}_{i=1}^{m},$$

where $p_t^{(i)}$ is the ith particle and $w_t^{(i)}$ its corresponding weight. In our splitting step, we further partition the data into S disjoint subproblems $E_t = \{E_t^1, E_t^2, \ldots, E_t^S\}$, with $\bigcup_{s=1}^{S} E_t^s = x_{1:t}$. Each subproblem maintains its own particle set

$$\mathcal{P}_t^s = \{\, p_t^{(s,i)} \}_{i=1}^{m_s}, \qquad \{\, w_t^{(s,i)} \}_{i=1}^{m_s},$$

where $p_t^{(s,i)}$ is a partition of only the observations in E_t^s, and $w_t^{(s,i)}$ is the corresponding weight. The maximum size of each subproblem particle set is fixed at a static value m, so the amount of information required to represent the stored clusterings is at most that of a vanilla particle filter with m particles, which is linear in dataset size. The number of weights required, however, is linear in the number of subproblems rather than fixed.

3.2 Split Step

Under a Dirichlet process prior, clusters are generated independently of each other. This means that if we condition on the event that two or more subsets of the data do not appear in clusters together, the true posterior distribution p based on the Dirichlet process mixture model factorises into independent subproblems (Ewens, 1972). We can use this property to decompose the clustering problem: suppose that our approximate posterior $\hat{p}$ partitions the data into sub-problems and assigns no probability to data points from different sub-problems being in the same cluster. Then we can replace it with a distribution $\hat{p}^M$, that calculates the marginal distribution under $\hat{p}$ for clusters for each sub-problem, and defines the joint distribution over clusterings such that clusterings for sub-problems are independent. Since we know that the true posterior probability factorises in this way for each of the particles that make up $\hat{p}$, the factorised $\hat{p}^M$ has better accuracy.

Proposition 3.1 (Informal statement)**.** *Let p denote the posterior distribution over clusterings of dataset*

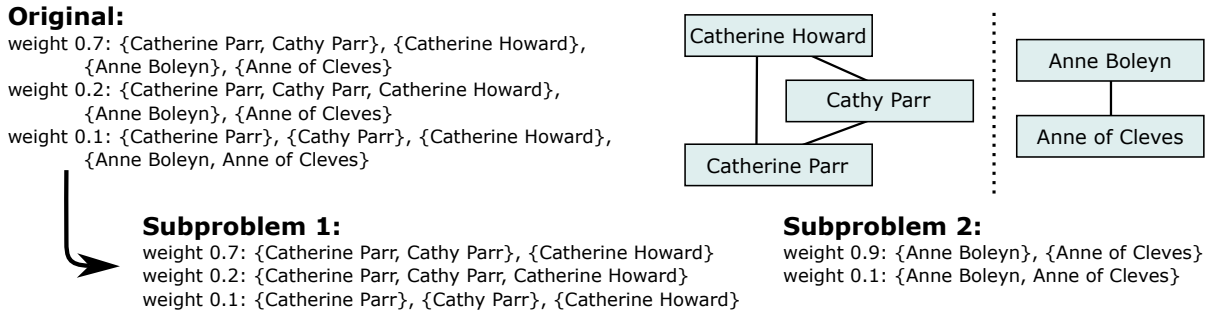

Figure 3: Splitting into subproblems based on particle filter output

X induced by a Dirichlet process mixture model. Suppose we have a distribution $\hat{p}$ over clusterings of X, and a partitioning of X into two or more disjoint subsets, such that $\hat{p}$ gives a probability of zero to any clustering where two datapoints in different partition elements appear in a cluster together. Let $\hat{p}^M$ be the distribution defined as the product of marginals of $\hat{p}$ for each partition element. Then $KL(\hat{p}^M||p) \leq KL(\hat{p}||p)$. Furthermore, $\hat{p}^M$ has the lowest KL divergence of any distribution with the same marginals as $\hat{p}$.

Proof. With formal statement in Appendix B.1 □

Splitting the clustering problem in this way allows us to implicitly represent a much larger particle set than could be represented explicitly with one particle per clustering of the full dataset, as the effective size of the new particle set is the product of the number of particles on each sub-problem (see Appendix A.2.1). This involves no extra storage cost compared to the original SMC approximation, nor does it involve any additional computational cost at the next iteration to update the approximation with a new observation, but is guaranteed to give a better approximation.

The no-overlap event can be detected by constructing a graph whose vertices are observations and whose edges connect any two points that co-occur in a cluster on at least one particle (see Figure 3). We can efficiently obtain a partition of the data into subproblems $\{E_t^1, \ldots, E_t^S\}$ by splitting this graph into its connected components (Pearce, 2005). Each component E_t^s induces a set of clusters

$$C_t^s \;=\; \big\{\, c \in \bigcup_{p \in \mathcal{P}_t} p : c \subseteq E_t^s \big\},$$

and hence a marginal particle set for its data,

$$\mathcal{P}_t^s \;=\; \{\, p \cap C_t^s \mid p \in \mathcal{P}_t \},$$
$$w_t^{(s,i)} = \sum_{j=1}^{|\mathcal{P}_t|} w_t^{(j)}\, \mathbb{1}[\, p_t^{(s,i)} \subseteq p_t^{(j)} \,]\,.$$

Collectively, the $\mathcal{P}_t^s$ represent the full posterior approximation as the product space $\bigotimes_{s=1}^{S} \mathcal{P}_t^s$, with factorised weights.

Figure 2 shows a visualisation of the resulting subproblem partitions on our 2D datasets. The subproblems are indicated by a bounding box drawn around their datapoints, as well as plotted with different markers. We highlight that the subproblem partition is decided dynamically by the posterior distribution rather than by a distance or similarity metric, since our algorithm aims to find the most accurate way of representing the joint clustering distribution with a limited amount of memory. As a result, the subproblems can be very close together in 2D space and may have nonlinear boundaries. Points that are nearest neighbours in a Euclidean sense are not necessarily placed in a subproblem together, since they may nevertheless have low posterior probability of being in the same cluster once the other datapoints are considered.

The splitting approach works best when the data have some degree of hierarchical structure: our algorithm makes splits when sub-populations of datapoints have greater within-population similarity than inter-population similarity, as the SMC sampler will discard inter-population clusterings before it discards within-population clusterings, and Proposition 3.1 will apply when this happens.

3.3 Update Step

In SMC clustering, when a new observation x_t arrives, we first generate a *putative* particle set by extending each previous particle $p_{t-1}^{(i)} \in \mathcal{P}_{t-1}$ in each possible way: (i) for each existing cluster $c \in p_{t-1}^{(i)}$ and (ii) for each new cluster $c = \varnothing$ (meaning x_t forms a singleton). We define

$$\tilde{p}_t(i,c) \;=\; \{c \cup_{\mathrm{m}} x_t\} \;\cup\; p_{t-1}^{(i)} \setminus \{c\},$$
$$\tilde{w}_t(i,c) \;=\; w_{t-1}^{(i)}\; p\big(x_t \text{ belongs to } c \mid p_{t-1}^{(i)}, x_{1:(t-1)}\big)\,, \tag{2}$$

where the weight update is given by (1). Collecting the $\{\tilde{p}_t(i,c), \tilde{w}_t(i,c)\}$ for each cluster on each particle yields the putative particle set.

After a split has been made, we modify this procedure to work with the factorised particle representation. Having split the data into subproblems $E^1_{t-1}, \ldots, E^S_{t-1}$, we can perform the same expansion independently on each subproblem s, forming a pooled set $\tilde{\mathcal{P}}_t$ of the putative particles $\tilde{p}_t(s,i,c)$ from each subproblem particle set, with weights $\tilde{w}_t(s,i,c)$ calculated as per (2). The particles from each subproblem with the *new cluster* assignment $c = \varnothing$ correspond to an identical global event, i.e. *x_t is a singleton cluster*. If we retained all such duplicates, we would overcount its weight. This can be avoided by only keeping the $\varnothing$ assignments from a single subproblem when forming $\tilde{\mathcal{P}}_t$, while discarding particles with $c = \varnothing$ from all other subproblems. We choose the subproblem with the highest weighted assignment for x_t.

After normalising the weights for the particle approximations $\mathcal{P}^s_{t-1}$ for each subproblem at time $t-1$, the particle approximation to the full posterior over clusterings of $x_{1:t}$ can be defined as follows. The probability of x_t being added to subproblem E^s_{t-1} and the clustering of this subproblem being $\tilde{p}_t(s,i,c)$ is proportional to $\tilde{w}_t(s,i,c)$; and then conditional on this, the clustering for subproblems $E^{s'}_{t-1}$, for $s' \neq s$, is drawn independently from the particle approximation $\mathcal{P}^{s'}_{t-1}$.

There are two problems with this approximation. First, it places x_t in all subproblems, violating the no-overlap condition that leads to independence across subproblems. Second, it involves more than m particles for the clustering of x_t. To deal with the latter, we perform resampling, so as to maintain the required number of particles at each iteration. Then, based on whether the resampled particles obey the no-overlap condition, we may need to merge sub-problems.

A greedy resample of the putative particles simply involves choosing the m values $\tilde{p}_t(s,i,c)$ that have largest weights $\tilde{w}_t(s,i,c)$. If the resampled particles all are within the same sub-problem, then our no-overlap condition holds with x_t allocated to the appropriate subproblem, and we replace its particle set with the resampled set. If not, we perform a merge across subproblems that include x_t as described in the next section. In either case, we then attempt the split step to further improve the partition into subproblems. We visualise this state update in Appendix C.

3.4 Merge Step

When the assignments for x_t span multiple subproblems, the data partition is updated by merging the affected subproblems, which we index by $\mathcal{S} \subseteq \{1, \ldots, S\}$. This is done by producing a sample from their joint distribution, using the fact that this was independent for assignments prior to x_t. The previous split particle set $\{\mathcal{P}^s_{t-1}\}_{s\in\mathcal{S}}$ implicitly represents a larger particle set of solutions to the full clustering problem on $x_{1:(t-1)}$, formed as a Cartesian product of the subproblem particles. Hence, each updated particle $\tilde{p}_t(s,i,c)$ implicitly represents a joint distribution over clusterings of $x_{1:t}$, where the clustering of x_t and the observations in E^s_{t-1} has been fixed. To form an explicit representation of this joint distribution, we combine $\tilde{p}_t(s,i,c)$ with every possible particle configuration p' drawn from the Cartesian product $\bigotimes_{j\in\mathcal{S}\setminus\{s\}} \mathcal{P}^j_{t-1}$. Then we have for each combination and weight

$$
\begin{aligned}
p_{\text{joint}} &= \tilde{p}_t(s,i,c) \;\cup\; p', \\
w_{\text{joint}} &= \tilde{w}_t(s,i,c) \;\times \prod_{j\in\mathcal{S}\setminus\{s\}} w^{(j,\cdot)}_{t-1}.
\end{aligned}
$$

Computing this expanded set for each $\tilde{p}_t(s,i,c)$ gives us an explicit form for the joint distribution represented by the output of the update step, across the subproblems in $\mathcal{S}$. Resampling this set yields a valid sample from the joint posterior over the affected subproblems.

Checking merges In this merge step, it is possible for a subproblem's assignments for x_t to be discarded entirely – in this case, the merge is undone by the next splitting step, with the existing particles on that subproblem thinned by the resample. We can improve the quality of the particle approximation by instead discarding particles where x_t is assigned to a given subproblem if they have sufficiently low weight (see Appendix B.2). A rule of thumb is to set a threshold of $1/m$ on their combined weight, since in this case the expected number of times assignments on that subproblem will be resampled in the merging step is less than one, and the merge is likely to be redundant.

Multinomial merging When $|\mathcal{S}|$ or the subproblem particle counts are large, performing merges by explicitly forming the full joint distribution can be expensive. A cheaper but higher-variance alternative is to use multinomial resampling in three stages: (i) sample m assignments of x_t (with replacement) from the marginal distribution over assignments, then (ii) for each sampled assignment, independently draw one particle from each of the other involved subproblems, and finally (iii) merge any duplicate particles.

3.5 Surrogate Models

In domains such as entity linking, accurate cluster likelihoods often depend on complex generative models whose evaluation cost grows with cluster size, which can become prohibitive when every potential assignment must be scored in an SMC update. To mitigate this, we introduce lightweight surrogate models that can quickly identify the most plausible cluster assignments. By using the resulting DPMM posterior under this surrogate likelihood as a proposal distribution within each update step (details in Appendix C), we concentrate neural likelihood evaluations on a smaller subset of high-probability assignments, drastically reducing model calls. We found that using a surrogate model in this way can even improve accuracy in the online setting (see Appendix A.3).

For continuous data with a meaningful distance metric, Gaussian mixture models can be used. For text data, we use character-level n-gram models with Dirichlet priors (Chien, 2015) to capture overlap patterns in entity names. Surrogate models can also be applied as proposal distributions in vanilla SMC and MCMC sampling.

4 EXPERIMENTS

We compare our method to a range of standard approaches on two 2D clustering problems and three text datasets. Our baselines are as follows:

- **Sequential Monte Carlo** (Fearnhead, 2004) - the online clustering algorithm described in Section 2.2, equivalent to our proposed algorithm without the split step.
- **Greedy clustering** - an online algorithm assigning each new datapoint to the cluster with the highest weight. This is a special case of SMC with one particle.
- **Gibbs sampler** (Neal, 2000) - an offline MCMC algorithm that iteratively resamples individual cluster assignments using Equation 1. Where a neural model is used, we also compare to a Metropolis-within-Gibbs version of this sampler, in which a surrogate model is used as a proposal distribution.
- **Agglomerative clustering** (Banfield and Raftery, 1993) - an offline algorithm that iteratively merges the pair of clusters with the highest value of the Bayesian hypothesis test statistic comparing the merged and un-merged clusterings under the DPMM. This terminates when no merges are found that pass the hypothesis test.

Evaluation is based on model log-posterior density, runtime, and clustering accuracy based on precision, recall and F1 scores (Bagga and Baldwin, 1998). The un-normalised log-posterior density is computed using Ewen's sampling formula (Equation 3 in Appendix B.1).

In MCMC inference, the sample with the highest posterior probability is used for this comparison. For the sequential Monte Carlo samplers, we use the highest-weighted particle in the particle set. The data were reshuffled for each replication in order to compare performance over different orderings. Offline methods were run until termination where possible, or for a maximum runtime of 10^4 seconds. We stopped the MCMC samplers when 500 sweeps passed without change to the estimated MAP clustering.

We present a summary of the clustering algorithms' runtime and performance on log-posterior density and F1 in Table 1. SMC results are reported for a fixed $m = 100$ particles. An extended comparison of the full results including precision and recall is presented in Appendix A, where we compare results for a wider range of hyperparameter configurations, including particle set size, to establish that our results are robust to changes in tuning and runtime budget. We give full experimental details in Appendix D, and full source code for our experiments is available on GitHub.

4.1 Circles

In this experiment, clusters of observations are generated as points on a circle of fixed radius, with each cluster having a different location. This problem is particularly challenging when clusters overlap with each other, since neighbouring points can belong to different clusters and more than two points must be considered jointly to determine the number and location of clusters. A dataset of 300 points across 15 clusters was generated, giving a problem where several groups of clusters had non-trivial overlap with each other, see Figure 2. A variational diffusion model (Kingma et al., 2021) was used to approximate cluster likelihoods as the model likelihood of observing a collection of datapoints of a given size, with a Gaussian mixture model used as a surrogate.

Results across 30 replications are reported in Table 1. The SMC samplers had equivalent or better performance on log-posterior density, recall, precision and F1 than the agglomerative algorithm. The MCMC sampler outperformed the sequential algorithms on log-posterior density and precision, at the expense of a runtime that was 10 times longer. Split SMC significantly outperformed vanilla SMC on all metrics, and had higher recall and F1 than the MCMC sampler.

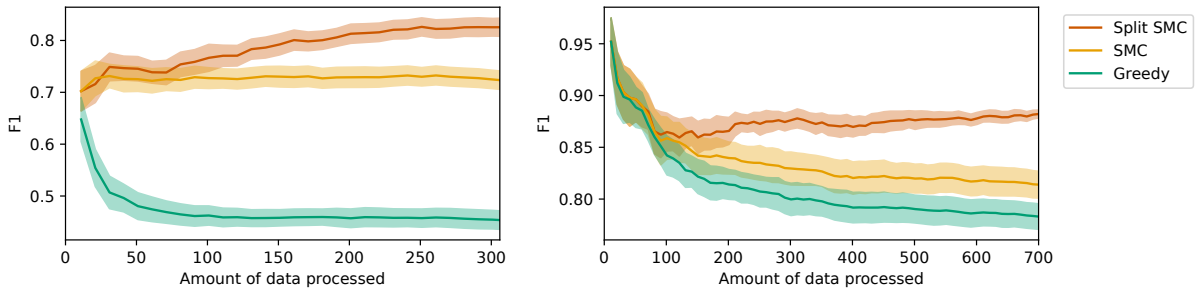

Figure 4: Online accuracy for the circles (left) and GMM (right) datasets. We show the average F1 score of the highest-weighted clustering as more data is observed, with shaded regions indicating 95% confidence intervals.

4.2 Gaussian Mixture

A dataset of 700 datapoints was generated from a Gaussian mixture model with a known distribution over cluster means and variances, plotted in Figure 2. Cluster assignments were generated from a Dirichlet process, resulting in 80 clusters with a skewed distribution of sizes.

Results across 20 replications are reported in Table 1. The split SMC algorithm had equivalent or better performance on log-posterior density, recall, precision, and F1 compared to the offline algorithms. Split SMC significantly outperformed vanilla SMC, and the variance of the split SMC output was lower. Despite initially having very similar maximum a posteriori clusterings, the samplers' performance diverged significantly as more data was observed (Figure 4). In contrast to the other sequential algorithms, the split SMC sampler's output had increasing accuracy and decreasing variance, showing a lower dependence on the order in which the data were observed. We highlight that the split SMC sampler's performance was equivalent to that of offline agglomerative clustering, despite the fact that agglomerative algorithms are particularly well suited to the Gaussian setting.

4.3 Knowledge Base Construction

We applied our algorithm to clustering datasets of entity fragments derived from the REBEL (Huguet Cabot and Navigli, 2021) and TweetNERD (Mishra et al., 2022) datasets, using a neural language model with a bigram surrogate (details in Appendix D.2.1). We used two datasets based on REBEL: REBEL-50 and REBEL-200, which had 50 and 200 clusters, and 428 and 1671 observations respectively. They contained observations of entity fragments with a range of attributes, collected from Wikipedia. The TweetNERD data had 200 clusters and 1931 fragments containing mentions of entity names extracted from tweets.

Both agglomerative clustering and MCMC failed to produce meaningful results on the larger datasets within our runtime budget of 10^4 seconds, so are omitted from these comparisons. We include a full comparison on the smaller REBEL-50 dataset, where they could be run to convergence.

Results are reported in Table 1, across 10 replications for REBEL-50 and 20 for the larger datasets. A GPU was used in both REBEL experiments, so the runtimes for REBEL are much shorter. The language model was trained on a training split of the original REBEL dataset and achieved much lower recall clustering TweetNERD, likely due to the greater variation in entity names. On REBEL-50, split SMC was outperformed only by the agglomerative algorithm on log-posterior density, but had the highest F1 of any method. On the larger datasets, split SMC outperformed the other online algorithms on log-posterior density, though we note that the gap between it and greedy clustering on this metric was not statistically significant at the 95% level on TweetNERD. Split SMC had the best F1 on REBEL-200, but was outperformed by vanilla SMC on TweetNERD.

5 DISCUSSION

Motivated by the knowledge base construction problem, we presented a novel sequential Monte Carlo algorithm for clustering and demonstrated its ability to scale to problems with complex cluster-likelihoods and a large number of clusters. Our split SMC algorithm detects and exploits near-independence in its approximation to the posterior distribution in order to produce a compact factorised representation of the algorithm state, and is therefore able to dynamically adjust the effective size of its own particle set without incurring prohibitive memory or computational cost.

We showed that for a fixed number of particles, split SMC is able to scale to clustering problems on which vanilla SMC struggles, achieving superior results on

Table 1: Summary of accuracy metrics for each clustering algorithm across the datasets. The mean value of each metric across replications is reported, with its standard deviation in brackets.

Method	Circles			Gaussian mixture			REBEL-50		
	Log posterior	F1	Runtime (minutes)	Log posterior	F1	Runtime (minutes)	Log posterior	F1	Runtime (minutes)
Greedy	-906 (175)	.46 (.06)	1.4	-1539 (34)	.78 (.03)	0.2	-8674 (27)	.66 (.01)	1.4
SMC	-202 (92)	.73 (.04)	1.6	-1497 (30)	.81 (.03)	0.4	-8791 (52)	.65 (.01)	1.4
Split SMC	10 (74)	**.82** (.05)	5.1	**-1424** (7)	**.88** (.01)	1.7	-8644 (17)	**.68** (.01)	1.6
MCMC	**187** (15)	.81 (.02)	54.7	-1428 (2)	**.88** (.01)	54.4	-8709 (19)	.63 (.01)	172.8
Agglomerative	-81 (48)	.73 (.04)	54.7	-1425 (0)	.87 (.00)	239.9	**-8543** (0)	.63 (.00)	172.2

Method	REBEL-200			TweetNERD		
	Log posterior	F1	Runtime (minutes)	Log posterior	F1	Runtime (minutes)
Greedy	-35382 (115)	.69 (.01)	6.6	-31780 (123)	.51 (.01)	163.7
SMC	-36096 (137)	.69 (.01)	18.5	-32923 (221)	**.56 (.01)**	51.5
Split SMC	**-35297** (108)	**.70** (.01)	11.0	**-31705** (110)	.55 (.01)	128.0
MCMC	–	–	–	–	–	–
Agglomerative	–	–	–	–	–	–

log-posterior density and, with the exception of the TweetNERD dataset, accuracy. Due to the larger number of unique clusterings represented by our algorithm and hence the larger number of model evaluations required, its runtime for a given particle set size is typically higher than that of vanilla SMC. We provide an extended analysis in Appendix A showing that our algorithm also outperforms vanilla SMC for a wide range of runtimes and particle set sizes.

Since most of our experiments required working with neural likelihoods, we used simple surrogate models to reduce the number of likelihood evaluations required. This allowed accurate clusterings to be found much more efficiently, and we found that using surrogate models in this way actually improved accuracy in the online setting (see Appendix A.3).

In addition, we found that split SMC was capable of achieving comparable results to offline methods despite operating in the online setting and requiring a substantially shorter runtime. In our experiments, it typically achieved equivalent or even superior results to the offline baselines, which in our larger experiments failed to converge or produce meaningful results within a reasonable runtime.

6 Acknowledgements

CT acknowledges the support of the EPSRC-funded EP/S022252/1 Centre for Doctoral Training in Statistics and Operational Research in Partnership with Industry (STOR-i), and Microsoft Research. PF was supported by EPSRC grants EP/Y028783/1, EP/R034710/1 and EP/R018561/1. CN was supported by EPSRC grants EP/Y028783/1 and EP/V022636/1 and the Research England Expanding Excellence in England (E3) funding for MARS: Mathematics for AI in Real-world Systems.

References

A. Bagga and B. Baldwin. Entity-based cross-document coreferencing using the vector space model. In *COLING 1998 Volume 1: The 17th International Conference on Computational Linguistics*, 1998.

J. D. Banfield and A. E. Raftery. Model-based Gaussian and non-Gaussian clustering. *Biometrics*, 49(3): 803–821, 1993.

O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *The VLDB Journal*, 18:255–276, 2009.

A. Bouchard-Côté, A. Doucet, and A. Roth. Particle Gibbs split-merge sampling for Bayesian inference in mixture models. *Journal of Machine Learning Research*, 18(28):1–39, 2017.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

K. Canini, L. Shi, and T. Griffiths. Online inference of topics with latent Dirichlet allocation. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 65–72. PMLR, 16–18 Apr 2009.

J. Carpenter, P. Clifford, and P. Fearnhead. An improved particle filter for non-linear problems. *IEE Proceedings Radar Sonar and Navigation*, 146(1):2–7, 1999.

J.-T. Chien. The shared Dirichlet priors for Bayesian language modeling. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2081–2085, 2015.

T. M. Cover. *Elements of Information Theory*. John Wiley & Sons, 1999.

A. Doucet, N. Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Statistics for Engineering and Information Science. Springer New York, New York, NY, 2001.

M. D. Escobar and M. West. Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association*, 90(430):577–588, 1995.

W. Ewens. The sampling theory of selectively neutral alleles. *Theoretical Population Biology*, 3(1):87–112, 1972.

P. Fearnhead. Particle filters for mixture models with an unknown number of components. *Statistics and Computing*, 14(1):11–21, 2004.

P. Fearnhead and P. Clifford. On-line inference for hidden Markov models via particle filters. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 65(4):887–899, 2003.

H. Ge, Y. Chen, M. Wan, and Z. Ghahramani. Distributed inference for Dirichlet process mixture models. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2276–2284, Lille, France, 07–09 Jul 2015. PMLR.

C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

J. Heek, A. Levskaya, A. Oliver, M. Ritter, B. Rondepierre, A. Steiner, and M. van Zee. Flax: A neural network library and ecosystem for JAX, 2023. URL `http://github.com/google/flax`.

K. A. Heller and Z. Ghahramani. Bayesian hierarchical clustering. In *Proceedings of the 22nd International Conference on Machine learning*, pages 297–304, 2005.

P.-L. Huguet Cabot and R. Navigli. REBEL: Relation Extraction By End-to-end Language generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*. Association for Computational Linguistics, Nov. 2021.

A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., USA, 1988. ISBN 013022278X.

N. Kantas, A. Doucet, S. S. Singh, J. Maciejowski, and N. Chopin. On particle methods for parameter estimation in state-space models. *Statistical Science*, 30(3):328 – 351, 2015.

D. P. Kingma, T. Salimans, B. Poole, and J. Ho. Variational diffusion models. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc.

T. Koo, F. Liu, and L. He. Automata-based constraints for language model decoding. In *First Conference on Language Modeling*, 2024.

I. Kruusamägi. Johannes Gutenberg (kass), 2024. CC BY-SA 4.0, via Wikimedia Commons. Cropped from original.

K. Kurihara, M. Welling, and Y. W. Teh. Collapsed variational Dirichlet process mixture models. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, page 2796–2801, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

J. Lee, A. Chen, Z. Dai, D. Dua, D. S. Sachan, M. Boratko, Y. Luan, S. M. R. Arnold, V. Perot, S. Dalmia, H. Hu, X. Lin, P. Pasupat, A. Amini, J. R. Cole, S. Riedel, I. Naim, M.-W. Chang, and K. Guu. Can long-context language models subsume retrieval, RAG, SQL, and more? *arXiv preprint 2406.13121*, 2024.

P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

N. G. Marchant, A. Kaplan, D. N. Elazar, B. I. P. Rubinstein, and R. C. Steorts. d-blink: Distributed end-to-end Bayesian entity resolution. *Journal of Computational and Graphical Statistics*, 30(2):406–421, 2021.

J. McAuliffe, D. Blei, and M. Jordan. Nonparametric empirical Bayes for the Dirichlet process mixture model. *Statistics and Computing*, 16:5–14, 03 2006.

S. Mishra, A. Saini, R. Makki, S. Mehta, A. Haghighi, and A. Mollahosseini. TweetNERD - end to end entity linking benchmark for tweets. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc.

K. P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023. August 2022 draft.

R. M. Neal. Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, 9(2):249–265, 2000.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. *PyTorch: an imperative style, high-performance deep learning library*. Curran Associates Inc., Red Hook, NY, USA, 2019.

D. J. Pearce. An improved algorithm for finding the strongly connected components of a directed graph. Technical report, Victoria University, Wellington, NZ, 2005.

M. D. Scherreik. Online clustering with Bayesian nonparametrics. 2020.

R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In K. Erk and N. A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, Aug. 2016. Association for Computational Linguistics.

Ö. Sevgili, A. Shelmanov, M. Arkhipov, A. Panchenko, and C. Biemann. Neural entity linking: A survey of models based on deep learning. *Semantic Web*, 13(3):527–570, 2022.

N. Shazeer. GLU variants improve transformer. *arXiv preprint 2002.05202*, 2020.

J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

P. Szekely, C. A. Knoblock, J. Slepicka, A. Philpot, A. Singh, C. Yin, D. Kapoor, P. Natarajan, D. Marcu, K. Knight, D. Stallard, S. S. Karunamoorthy, R. Bojanapalli, S. Minton, B. Amanatullah, T. Hughes, M. Tamayo, D. Flynt, R. Artiss, S.-F. Chang, T. Chen, G. Hiebel, and L. Ferreira. Building and using a knowledge graph to combat human trafficking. In *The Semantic Web - ISWC 2015*, pages 205–221, Cham, 2015. Springer International Publishing.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

X. Wang, T. Isazawa, L. Mikaelyan, and J. Hensman. KBLam: Knowledge base augmented language model. In *The Thirteenth International Conference on Learning Representations*, 2025.

G. Weikum and M. Theobald. From information to knowledge: harvesting entities and relationships from web sources. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, page 65–76, New York, NY, USA, 2010. Association for Computing Machinery.

Wikimedia Foundation. Wikidata, 2012. URL `https://www.wikidata.org`.

J. Winn, M. Venanzi, T. Minka, I. Korostelev, J. Guiver, E. Pochernina, P. Mishkov, A. Spengler, D. Wilkins, S. Lindley, R. Banks, S. Webster, and Y. Zaykov. Enterprise Alexandria: Online high-precision enterprise knowledge base construction with typed entities. In *3rd Conference on Automated Knowledge Base Construction*, 2021.

K. Zaporojets, L.-A. Kaffee, J. Deleu, T. Demeester, C. Develder, and I. Augenstein. TempEL: Linking dynamically evolving and newly emerging entities. In

S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 1850–1866. Curran Associates, Inc., 2022.

B. Zhang and R. Sennrich. Root mean square layer normalization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes/No/Not Applicable]

   Mathematical setting given in Section 2.1. Algorithm details given in Section 3 and Appendix D.1. Assumptions for theoretical results given in Appendix B. Model details given in Appendix D.2.

   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes/No/Not Applicable]

   Memory and runtime costs are discussed in Section 3.1 and Appendix A.1.

   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes/No/Not Applicable]

   A package implementing our algorithm is available on GitHub at https://github.com/microsoft/smc-clustering.

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes/No/Not Applicable]
   (b) Complete proofs of all theoretical results. [Yes/No/Not Applicable]
   (c) Clear explanations of any assumptions. [Yes/No/Not Applicable]

   Proofs and assumptions can be found in Appendix B.

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes/No/Not Applicable]

   The experiment source code is available on GitHub. We give experimental details in Appendix D, describing the data used in Appendix D.2.

   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes/No/Not Applicable]

   Details in Appendix D.2.

   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes/No/Not Applicable]

   These are given in the table and figure captions.

   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes/No/Not Applicable]

   This can be found in Appendix D.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes/No/Not Applicable]
   (b) The license information of the assets, if applicable. [Yes/No/Not Applicable]

   See Appendix D.2.1.

   (c) New assets either in the supplemental material or as a URL, if applicable. [Yes/No/Not Applicable]
   (d) Information about consent from data providers/curators. [Yes/No/Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Yes/No/Not Applicable]

REBEL (Huguet Cabot and Navigli, 2021) and TweetNERD (Mishra et al., 2022) potentially contain personally identifiable information. Their contents are entirely drawn from text publicly available on Wikipedia and Twitter respectively.

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Yes/No/Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Yes/No/Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Yes/No/Not Applicable]

Scalable Model-Based Clustering with Sequential Monte Carlo: Supplementary Materials

A ADDITIONAL RESULTS

A.1 Full Results and Runtime Comparison

Here, we present the full results of each clustering algorithm on our experiments. Figures 5 to 9 plot each metric against the runtime of each algorithm. For the offline methods, we plot the metrics every 10 iterations. We also compare to faster, approximate agglomerative algorithms that use subsampling (details in Section D.1) and note their batch size in parentheses. The sequential methods were run for different particle set sizes (or model evaluation budgets in the case of greedy clustering) to measure performance across a range of possible runtimes.

The star markers in the figures indicate the algorithm configurations reported in the main paper – offline methods were run as close to convergence as possible (see Section D.1), while SMC methods used a fixed particle set size of $m = 100$. On the circles data, the Metropolis-within-Gibbs MCMC sampler achieved slightly superior results to exact Gibbs in a shorter runtime, so this is the result reported in the main paper for MCMC.

Section A.2 additionally presents the SMC results against the number of particles used. Our split SMC algorithm typically has a higher runtime than vanilla SMC for a fixed number of particles, primarily because it requires more model evaluations per iteration, since it stores a larger number of unique clusterings with the same number of particles. The splitting step also introduces a small additional computational overhead. We reported results for a fixed $m = 100$ in the main paper for the sake of consistency across experiments, as the runtime difference was heavily dependent on the computational resources available in each experiment. In addition, fixing m highlighted that our algorithm is not typically particularly sensitive to the choice of this hyperparameter, since the effective number of particles is adjusted dynamically depending on the problem's characteristics.

While algorithm runtimes are highly implementation-dependent, we present them to give an idea of how the various algorithm computational costs interact in practical settings – for example: neural model evaluations, surrogate model evaluations, parallelism, and computational overhead from state updates. Agglomerative methods tend to require many more model evaluations than other algorithms, while Metropolis-within-Gibbs requires far fewer (only 1-2 per update, to compute an acceptance probability). However, the model evaluations in agglomerative and SMC clustering can be run in parallel batches at each iteration, while Gibbs updates are inherently sequential, leading to much longer runtimes for the same number of model evaluations. This difference can be seen in the REBEL experiments, which were run on a GPU - the runtime difference between SMC and split SMC is very small despite split SMC using more model evaluations, because the additional evaluations can be run in parallel.

In all experiments, it is possible to achieve reasonable accuracy with sequential methods in a much shorter time than with any offline method considered. In addition, there is a band of runtimes for which split SMC has equivalent or better results compared to every other method, on every metric except precision. Since the offline methods are initialised with every observation in a singleton cluster (giving a precision of 1), they have high precision at short runtimes. Vanilla SMC also has higher precision than split SMC on the text data, but lower recall.

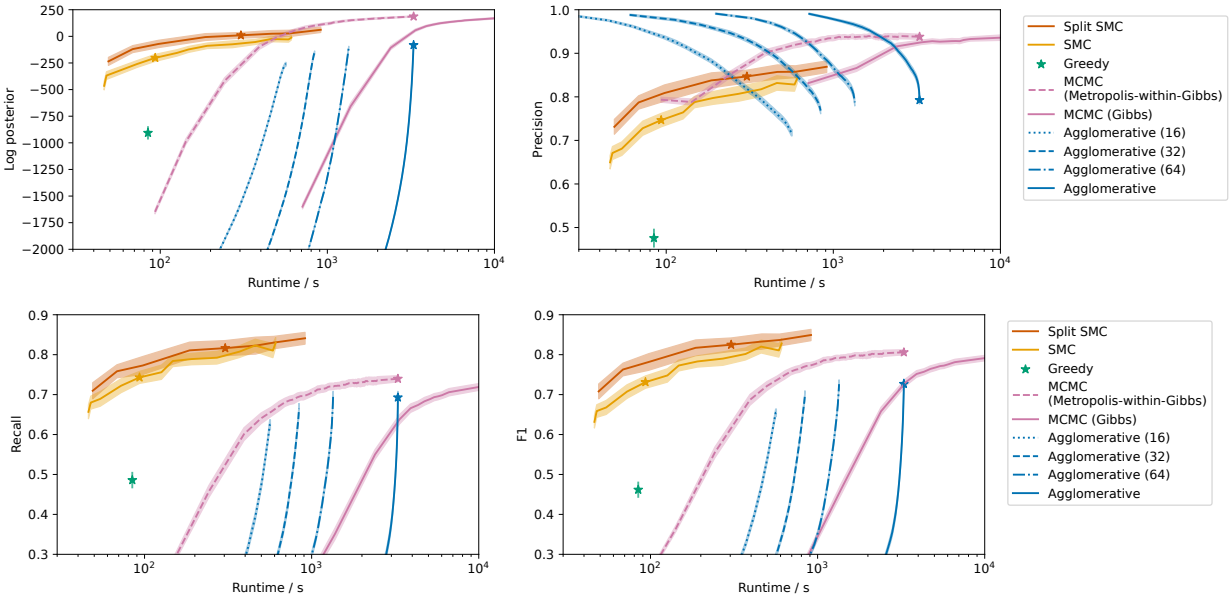

Figure 5: Performance metrics against runtime (log scale) for the circles dataset. The mean across replications is plotted, with a shaded region indicating a 95% confidence interval. The stars show the configurations reported in Table 1.

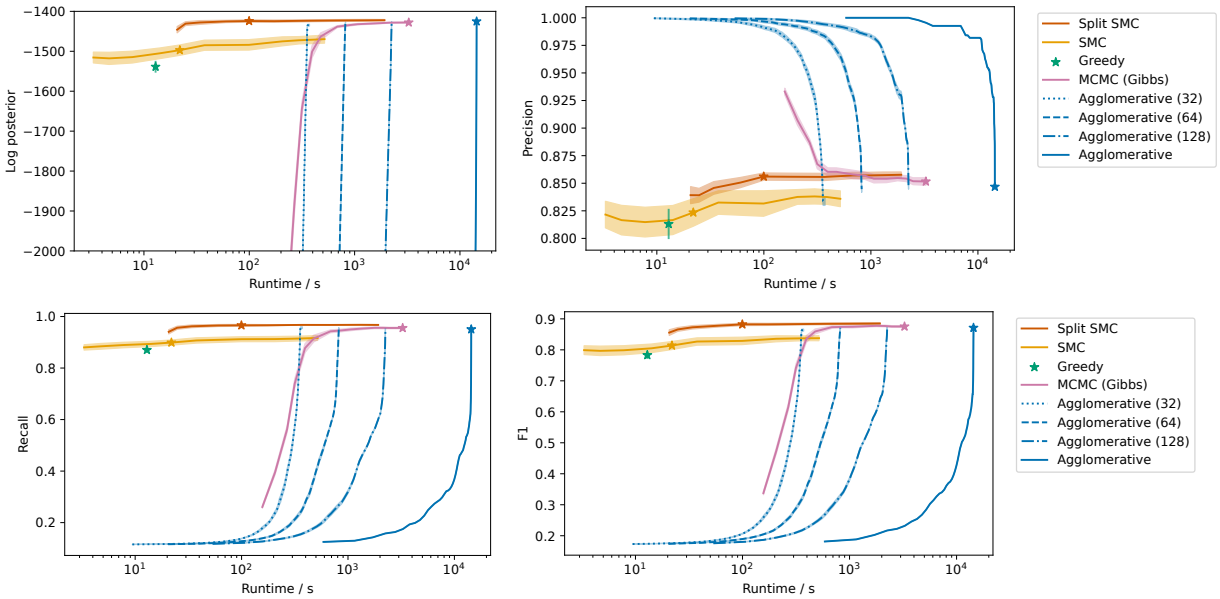

Figure 6: Performance metrics against runtime (log scale) for the GMM dataset. The mean across replications is plotted, with a shaded region indicating a 95% confidence interval. The stars show the configurations reported in Table 1.

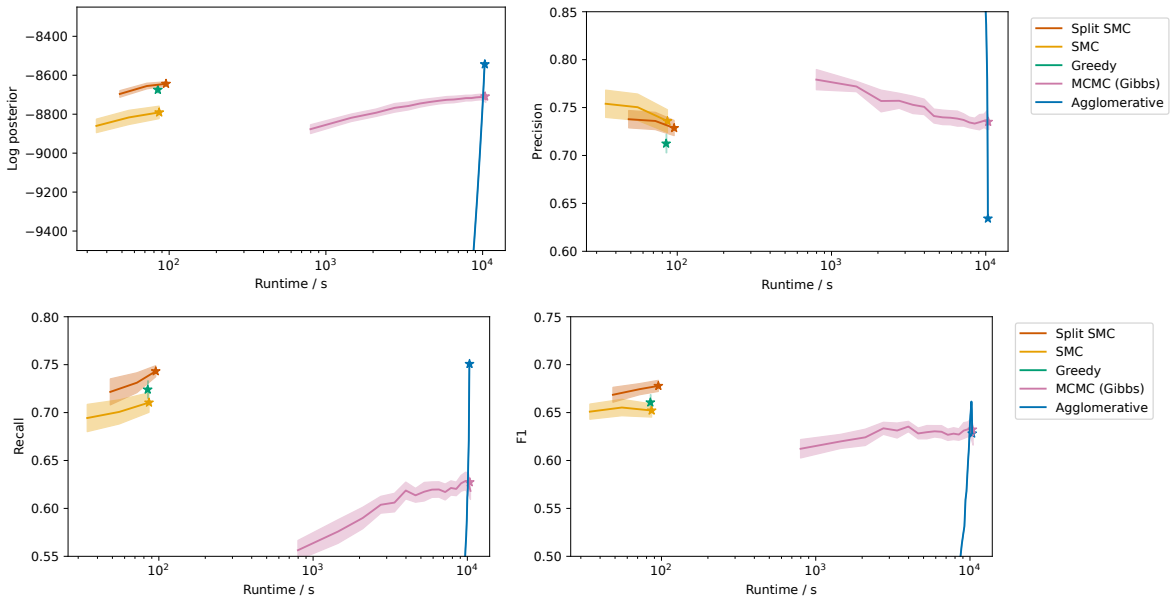

Figure 7: Performance metrics against runtime (log scale) for the REBEL-50 dataset. The mean across replications is plotted, with a shaded region indicating a 95% confidence interval. The stars show the configurations reported in Table 1.

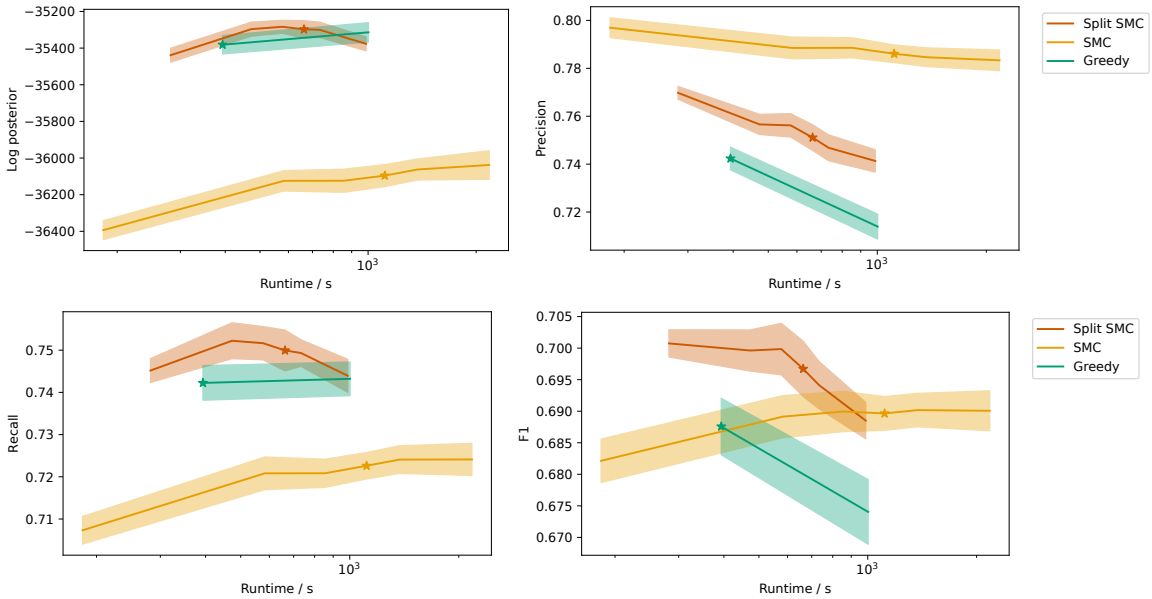

Figure 8: Performance metrics against runtime (log scale) for the REBEL-200 dataset. The mean across replications is plotted, with a shaded region indicating a 95% confidence interval. The stars show the configurations reported in Table 1.

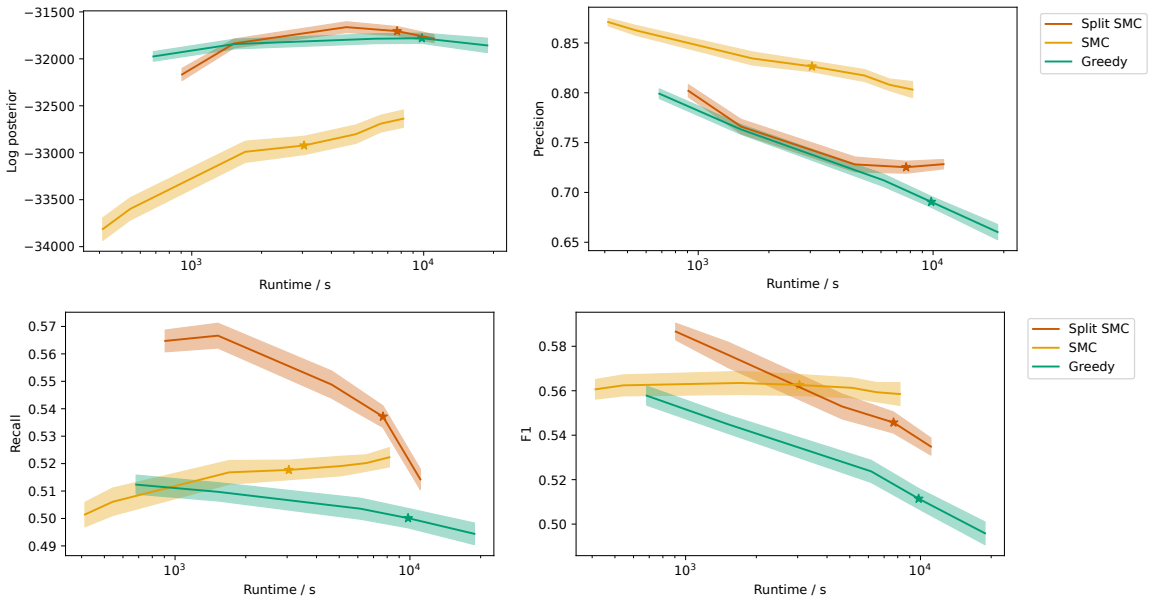

Figure 9: Performance metrics against runtime (log scale) for the TweetNERD dataset. The mean across replications is plotted, with a shaded region indicating a 95% confidence interval. The stars show the configurations reported in Table 1.

A.2 Particle Set Size Comparison

Figures 10 to 14 plot each metric against particle set size for the SMC algorithms. When using neural likelihood models on CPU, split SMC had a higher runtime than vanilla SMC for any given particle set size m, so we ran vanilla SMC for a larger range of m to enable a fair comparison on runtime.

A.2.1 Effective Particle Set Size in Split SMC

The number of particles implicitly represented by the split particle set can be computed by taking the product of the sizes of each subproblem. We plot the number of subproblems and the log of the effective number of particles for each experiment in Figure 15. The number of subproblems is decreasing in m in each case, since larger values of m allow us to store lower-probability clusterings that introduce overlap between potential partitions of the dataset. This decrease is dramatic for smaller values of m, but levels off as m increases. The effective number of particles typically increases with m regardless. However, on the text datasets, it decreases again for larger values: this may explain why increasing m past 100 did not improve the log-posterior density in these experiments.

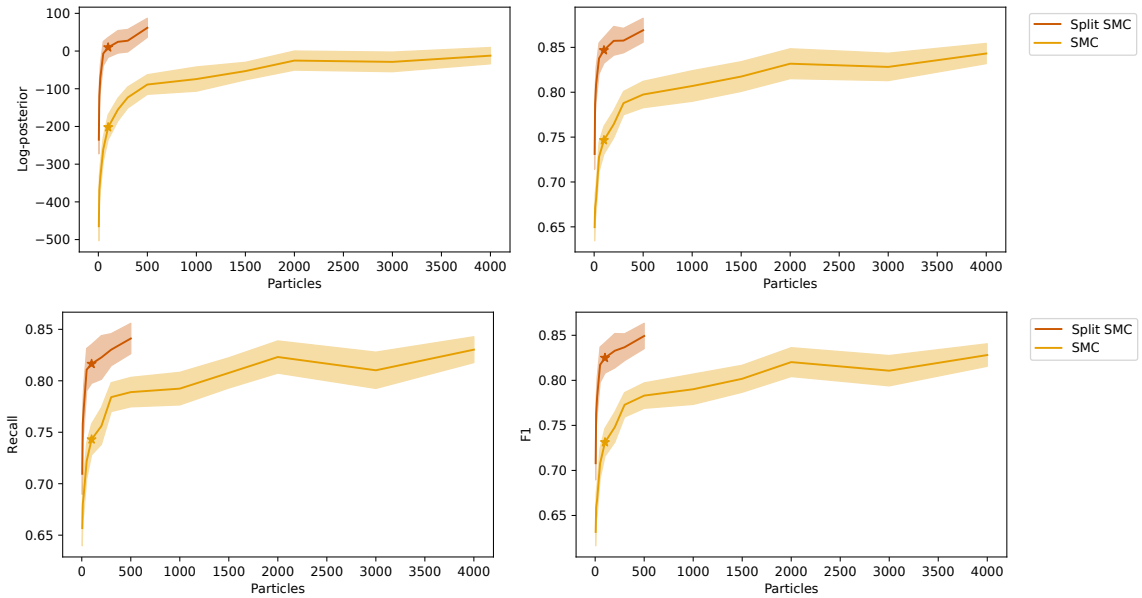

Figure 10: Performance metrics against particle set size for the circles dataset. The mean across replications is plotted, with a shaded region indicating a 95% confidence interval. The stars show the configurations reported in Table 1.

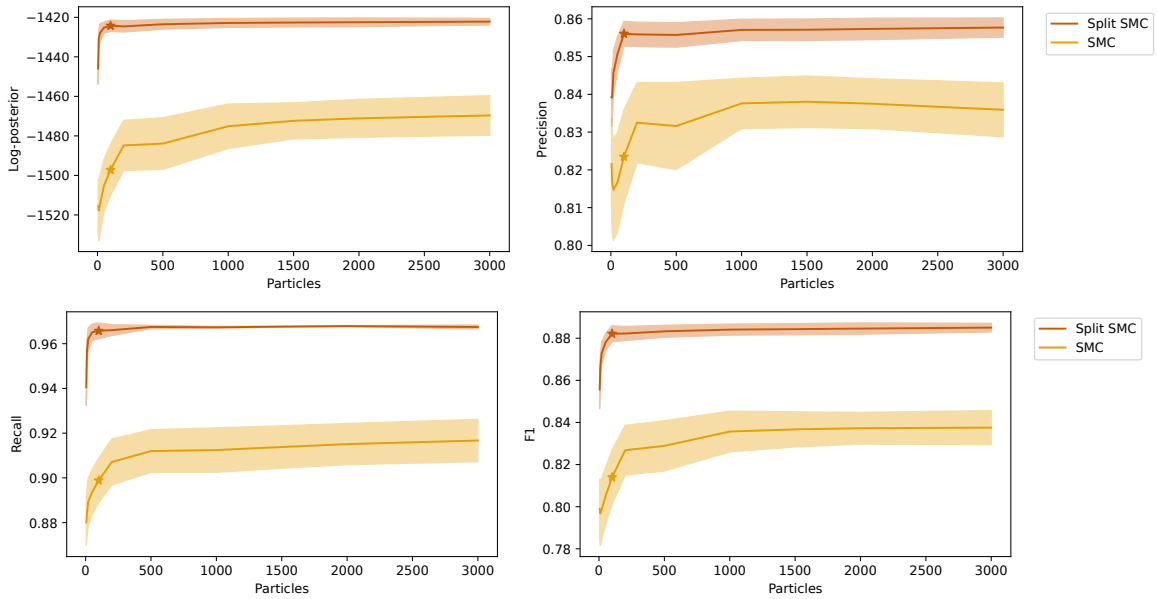

Figure 11: Performance metrics against particle set size for the GMM dataset. The mean across replications is plotted, with a shaded region indicating a 95% confidence interval. The stars show the configurations reported in Table 1.

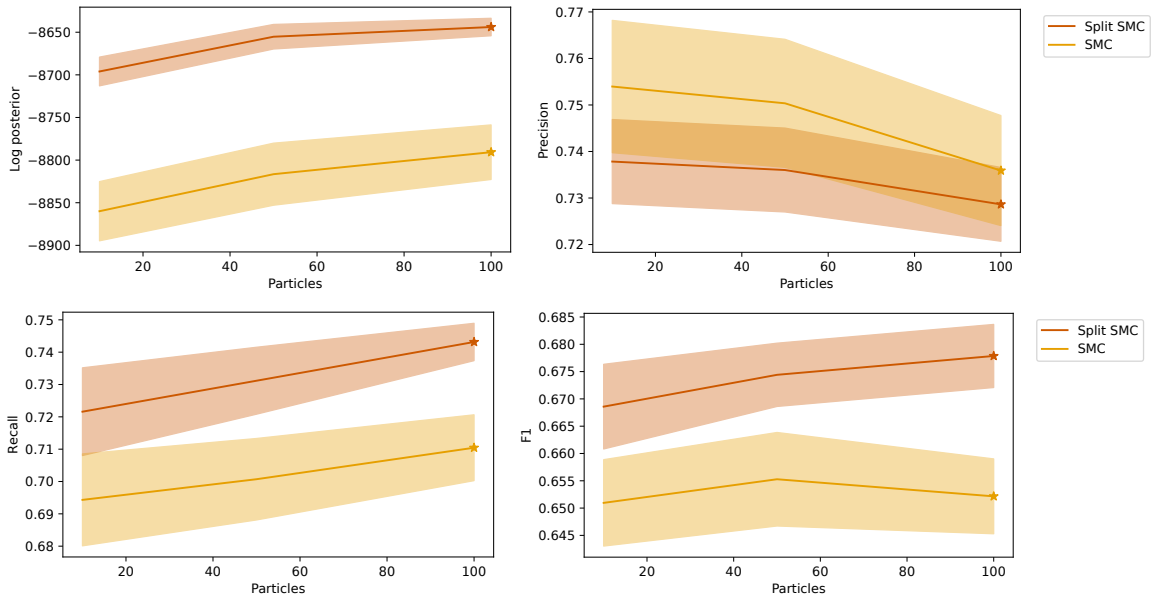

Figure 12: Performance metrics against particle set size for the REBEL-50 dataset. The mean across replications is plotted, with a shaded region indicating a 95% confidence interval. The stars show the configurations reported in Table 1.

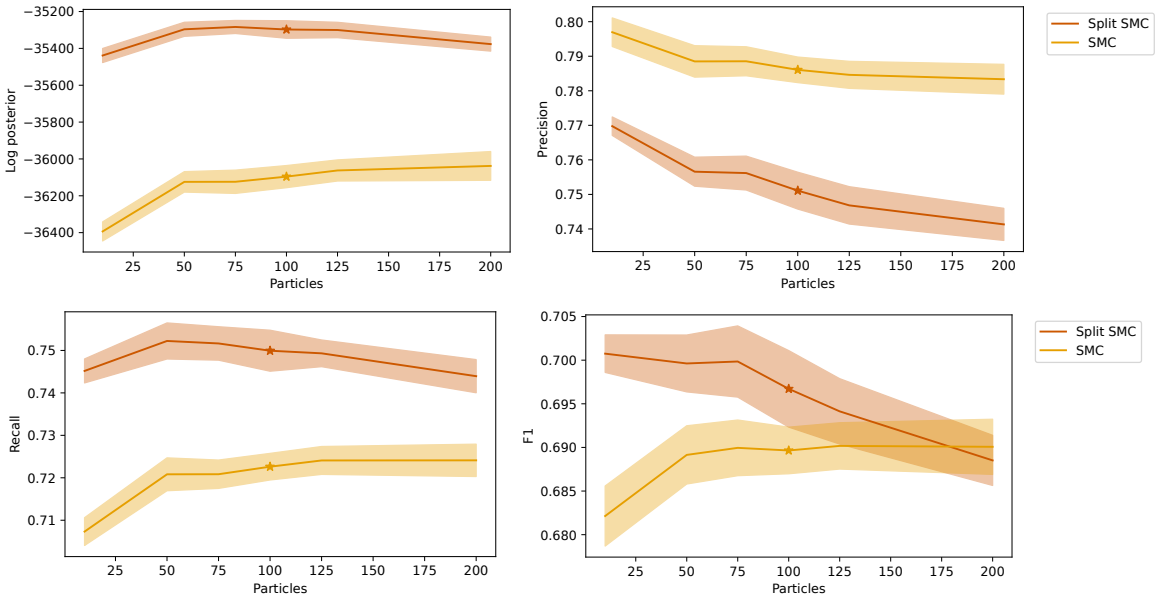

Figure 13: Performance metrics against particle set size for the REBEL-200 dataset. The mean across replications is plotted, with a shaded region indicating a 95% confidence interval. The stars show the configurations reported in Table 1.

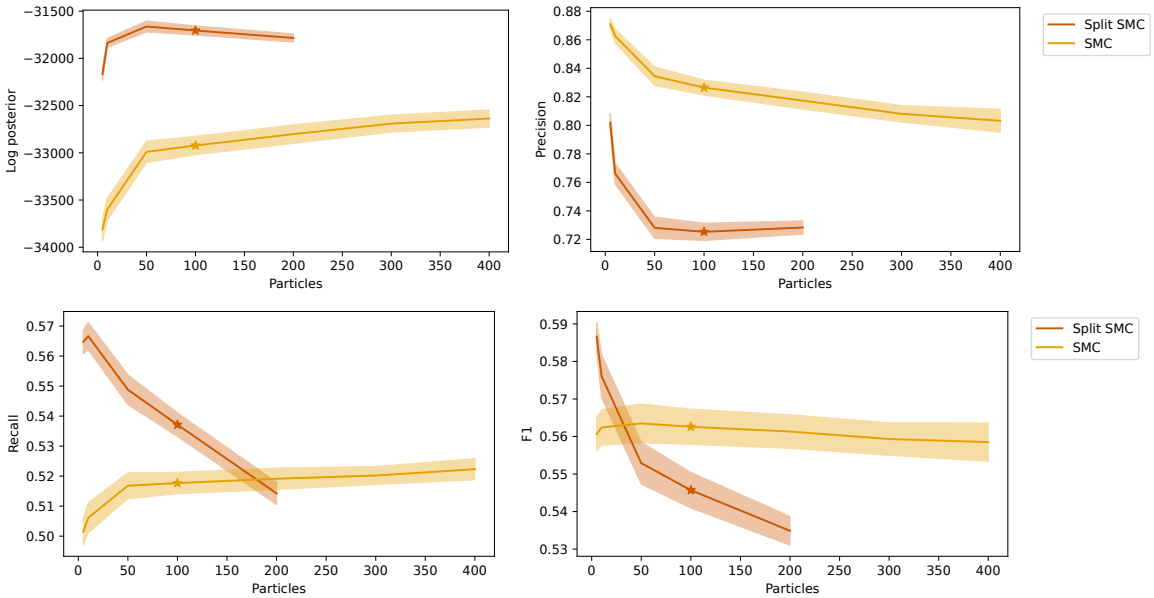

Figure 14: Performance metrics against particle set size for the TweetNERD dataset. The mean across replications is plotted, with a shaded region indicating a 95% confidence interval. The stars show the configurations reported in Table 1.

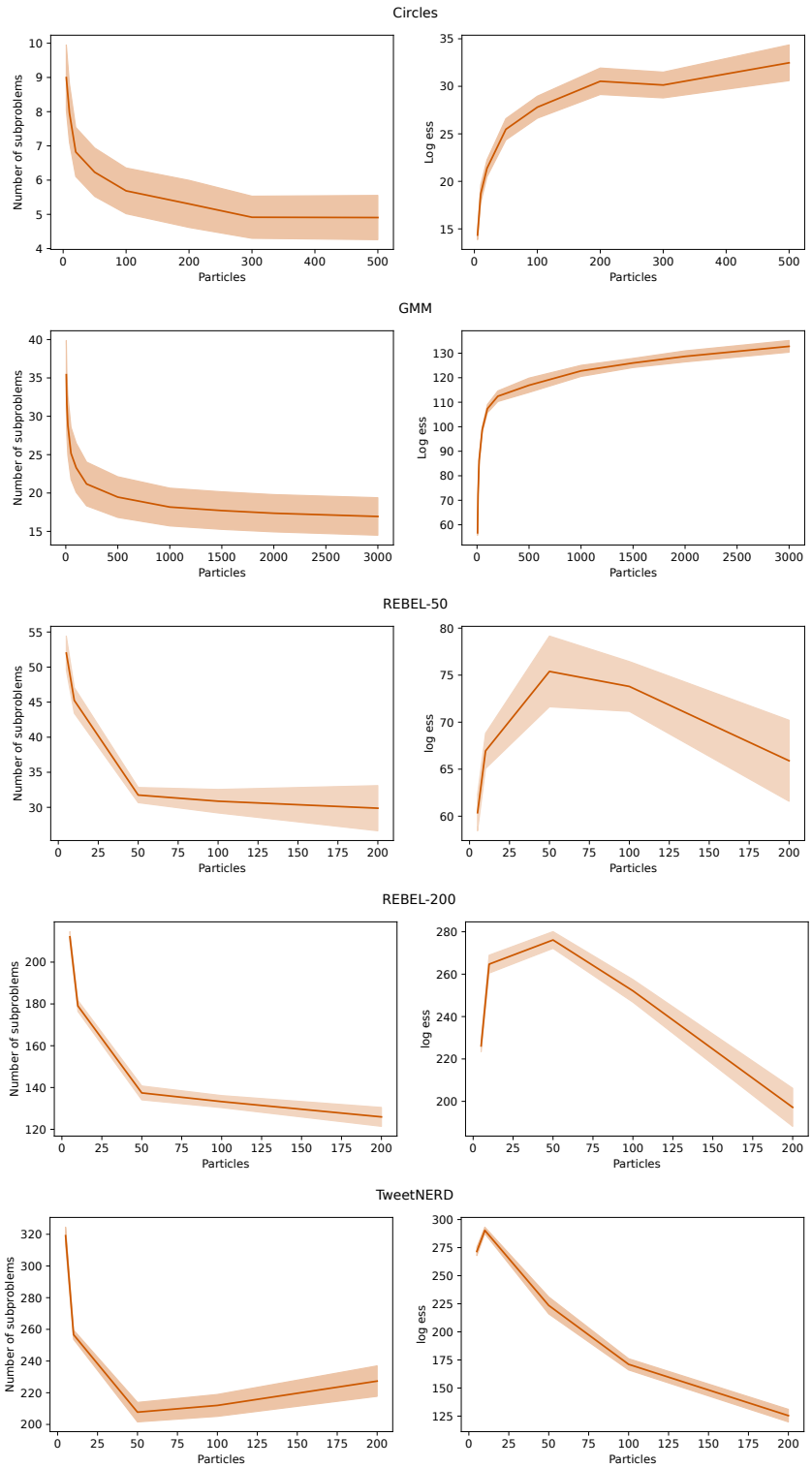

Figure 15: Number of subproblems (left) and log effective particle set size (right) against particle set size m for each dataset. The mean across replications is plotted, with a shaded region indicating a 95% confidence interval.

A.3 Surrogate Model Ablation

We investigate the impact of using surrogate models by fixing the number of particles used in sequential clustering and varying the number of model evaluations made. This was done by changing the number of particles proposed under the surrogate likelihood, which are then re-weighted and resampled according to the model likelihood. We call this parameter m', and a detailed explanation of the surrogate proposal step can be found in Appendix C. In the limit as the parameter m' increases, the surrogate model is not used at all. We used $m' = m$ in all previous experiments, with the exception of the greedy algorithm where this was set according to evaluation budget (not used for 2D data, and $m' = 100$ reported for text data). In each of the experiments below, we found that the clustering with the highest F1 was found by split SMC with $m' = m$, and that split SMC found the clusterings with the highest log-posterior density for each m'.

Figure 16 shows results for $m = 50$ on the circles data. Surprisingly, increasing the number of model evaluations actually decreases the log-posterior density of the clusterings found for each algorithm, and has a negative impact on every accuracy metric. The dashed horizontal lines on the plot indicate results obtained with $m = 50$ using the surrogate model alone – we can see that the surrogate model results have much higher recall but lower precision compared to those obtained with the neural likelihood model. The F1 obtained with the surrogate is lower than that of split SMC with $m' = m = 50$, but greater than or equal to those of every other configuration.

Figures 17 to 19 show the results for $m = 10$ on the REBEL-50, REBEL-200 and TweetNERD data. This time, increasing the number of model evaluations increases log-posterior density, indicating a greater mismatch between the surrogate and neural models. On TweetNERD, log-posterior density starts to decline again as m' increases. On both REBEL datasets, precision decreases and recall increases in m' for all methods. On REBEL-200, F1 decreases with m' for split SMC and greedy, while for vanilla SMC it increases a little and then starts to decline again slowly.

On TweetNERD, accuracy metrics are decreasing in m' for all methods, despite improvements in log-posterior density. This mirrors the results in Figure 14, where the accuracy of split SMC is decreasing in the number of particles. This may be due to the fact that, on this problem, the surrogate model produced more accurate clusterings when used alone than when the neural model was used. This is likely because the TweetNERD dataset was very out-of-domain for the model, which was trained on REBEL. TweetNERD contained fragments with the name attribute only, and these observations had considerably more within-cluster variability than the REBEL training data. Nevertheless, increased use of the surrogate model allowed algorithms to find clusterings with similar log-posterior density but higher accuracy.

These results indicate that using a surrogate model not only helps reduce runtime by limiting model evaluations, but is also beneficial for accuracy and can even help find clusterings with higher model likelihood in some problems. This effect was not observed with MCMC: Figure 5 shows that on the circles data, while there is a substantial improvement in runtime, there is minimal difference between results obtained with normal Gibbs sampling and Metropolis-within-Gibbs, where the surrogate is similarly used as a proposal distribution to reduce model evaluations. This may be due to the challenges of clustering with a complex likelihood function in the online setting – over-optimising model likelihood early in clustering may cause algorithms to discard more accurate clusterings that would later have higher weight after more data is observed. Combining a neural likelihood with a good choice of surrogate model likely has a regularising effect, as clusterings with low surrogate likelihood are discarded by the proposal step regardless of their likelihood under the model. The surrogate models used had very good recall, and seemed to discard correct clusterings relatively rarely.

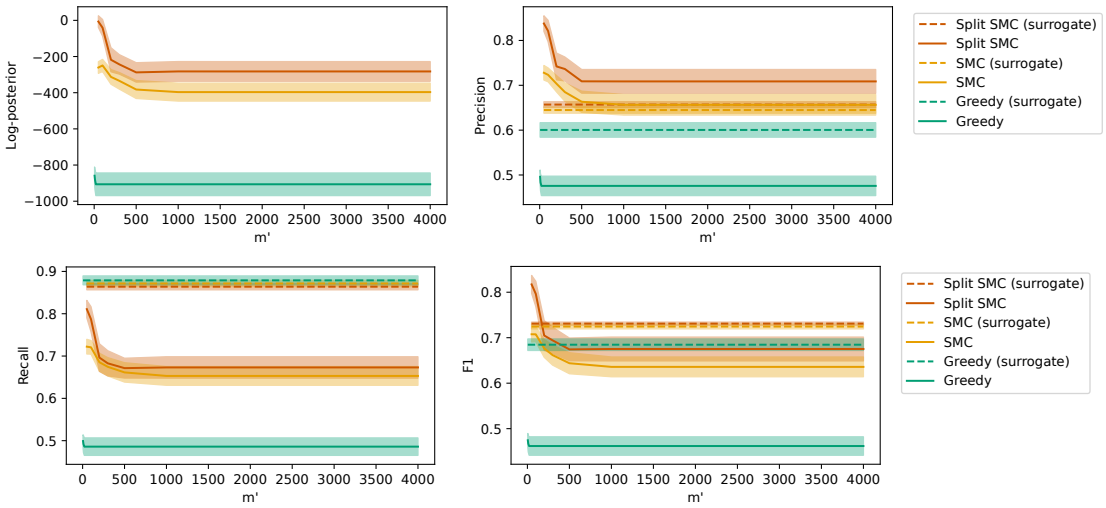

Figure 16: Performance metrics against surrogate proposal size m' on the circles dataset. The mean across replications is plotted, with a shaded region indicating a 95% confidence interval. Dashed horizontal lines indicate results using the surrogate model alone.

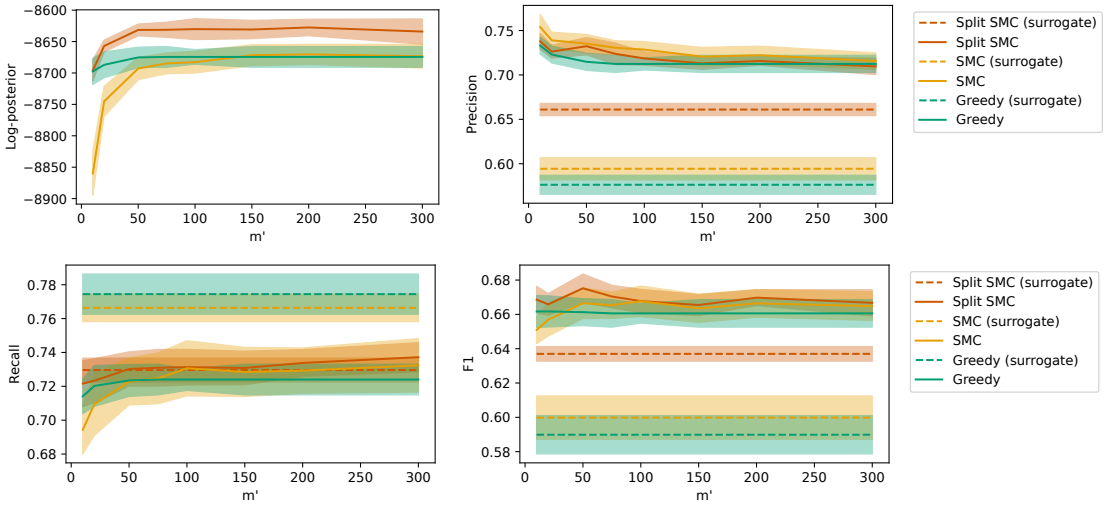

Figure 17: Performance metrics against surrogate proposal size m' on the REBEL-50 dataset. The mean across replications is plotted, with a shaded region indicating a 95% confidence interval. Dashed horizontal lines indicate results using the surrogate model alone.

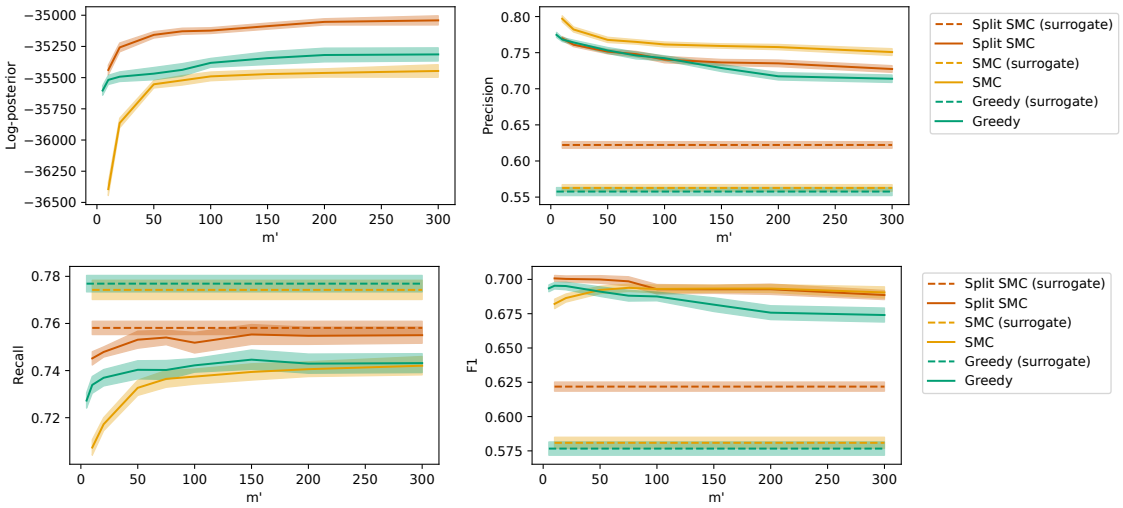

Figure 18: Performance metrics against surrogate proposal size m' on the REBEL-200 dataset. The mean across replications is plotted, with a shaded region indicating a 95% confidence interval. Dashed horizontal lines indicate results using the surrogate model alone.

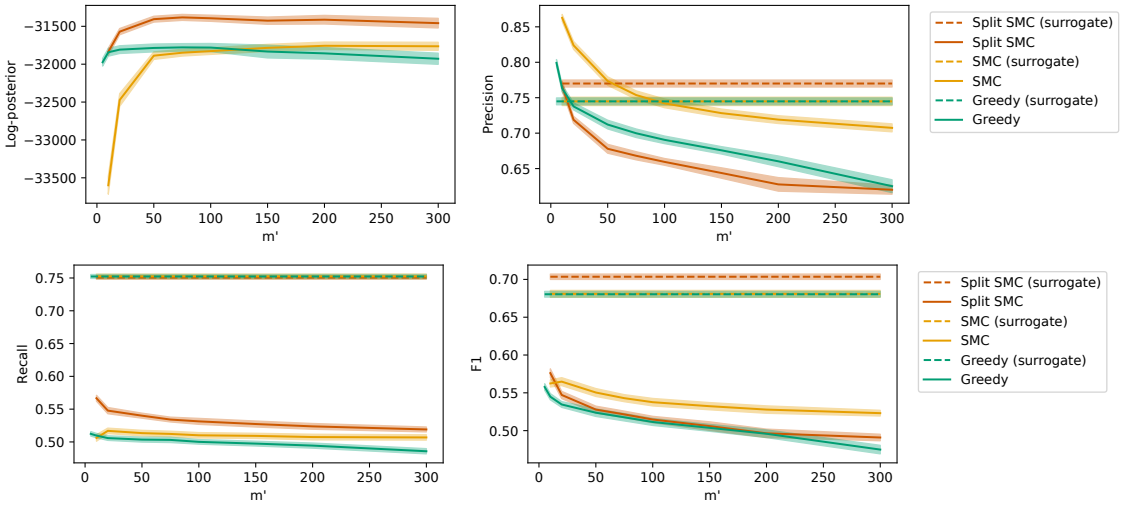

Figure 19: Performance metrics against surrogate proposal size m' on the TweetNERD dataset. The mean across replications is plotted, with a shaded region indicating a 95% confidence interval. Dashed horizontal lines indicate results using the surrogate model alone.

B PROOFS

B.1 Posterior Factorisation

We restate and prove a more formal version of Proposition 3.1, motivating the splitting approach used in our split SMC algorithm.

By Ewens's sampling formula (Ewens, 1972), the posterior probability of a dataset $X = x_{1:t}$ having the cluster labels $z_{1:t}$ can be written as

$$p(z_{1:t} \mid x_{1:t}) \;\propto\; \underbrace{\frac{\alpha^{K-1}}{\prod_{i=0}^{t-1}(\alpha+1+i)}\Big(\prod_{k=1}^{K}\Gamma(|c_k|)\Big)}_{\text{prior}}\;\underbrace{\Big(\prod_{k=1}^{K}p(c_k)\Big)}_{\text{likelihood}}. \tag{3}$$

Here, c_k refers to the collection of datapoints given cluster label k, $|c_k|$ is the number of datapoints in cluster k, and $p(c_k)$ refers to the likelihood of observing the given unordered collection in a sample of that size. It can be obtained by integrating out some unknown parameter θ, conditioned on which the datapoints within a cluster are independent of each other.

Suppose we partition the dataset X into disjoint subsets E^1 and E^2. For a given clustering, if none of the clusters c_k contains elements from both E^1 and E^2, and without loss of generality we relabel the clusters so the first J contain elements of E^1 and the last $K-J$ contain elements of E^2, then the posterior density factorises:

$$p(z_{1:t} \mid x_{1:t}) \;\propto\; \Big(\alpha^{J-1}\prod_{k=1}^{J}\Gamma(|c_k|)\,p(c_k)\Big) \;\times\; \Big(\alpha^{K-J-1}\prod_{k=J+1}^{K}\Gamma(|c_k|)\,p(c_k)\Big),$$

where the first product runs over the clusters contained in E^1 and the second over those in E^2. Conditioning on the event that no cross-subset clusters occur thus yields

$$p(z_{1:t} \mid x_{1:t}, \text{no-overlap}) \;=\; p(z_{E^1} \mid x_{E^1}, \text{no-overlap}) \;\times\; p(z_{E^2} \mid x_{E^2}, \text{no-overlap}).$$

Since the true posterior factorises for any clustering consistent with a given partition, the following result applies:

Proposition B.1. *Suppose we have a partition of the dataset X into two elements E^1 and E^2. We index possible clusterings of E^1 by $i = 1,\dots,I$ and clusterings of E^2 by $j = 1,\dots,J$, writing p_{ij} for the true posterior probability of clustering (i,j). Let p factorise so that $p_{ij} = p_i^1 p_j^2 \;\; \forall (i,j)$. Suppose we have a distribution $\hat{p}$ with support$(\hat{p}) \subseteq \{1,\dots,I\}\times\{1,\dots,J\} \subseteq$ support(p) and marginal distributions $\{b_i\}$ and $\{c_j\}$ over clusterings of E^1 and E^2. Then, the distribution q with marginals $\{b_i\}$ and $\{c_j\}$ that minimizes $KL(q\|p)$ is the product of these marginals, $q_{ij} = c_i b_j \;\forall i,j$.*

Proof.

$$\begin{aligned}
\mathrm{KL}(q\|p) &= \sum_{i,j} q_{ij}\log\frac{q_{ij}}{p_{ij}} = \sum_{i,j}\left(q_{ij}\log q_{ij} - q_{ij}\log p_{ij}\right)\\
&= \sum_{i,j}\left(q_{ij}\log q_{ij} - q_{ij}\log p_i^1 - q_{ij}\log p_j^2\right)\\
&= \sum_{i,j} q_{ij}\log q_{ij} - \sum_i \log(p_i^1)\sum_j q_{ij} - \sum_j \log(p_j^2)\sum_i q_{ij}\\
&= \underbrace{\sum_{i,j} q_{ij}\log q_{ij}}_{-H(q)} - \underbrace{\sum_i b_i\log p_i^1 - \sum_j c_j\log p_j^2}_{\text{constant}}
\end{aligned}$$

The first term is the negative Shannon entropy H of q, whilst the second depends only on the marginals of p and q, which are fixed. Since q is a coupling of b and c, we have that $H(q) \le H(b)+H(c)$, with equality achieved when

$q = b \otimes c$ (Cover, 1999). Hence, taking q to be the product of the marginals of $\hat{p}$ minimises KL divergence. Note that we take the reverse KL here, since for a particle approximation we typically have support$(\hat{p}) \subsetneq$ support(p) and hence the forward KL, $\mathrm{KL}(p||\hat{p})$, is infinite. □

By induction, this result also holds for partitions into more than two elements such that $\hat{p}$ gives a probability of zero to any clustering where two datapoints on different partition elements are in a cluster together.

B.2 Greedy Resampling

In this section, we provide some theoretical motivation for using greedy resampling in SMC, and for discarding particles that introduce overlap in the split representation.

Proposition B.2. *Let x and y be discrete probability distributions with $\mathcal{Y} :=$ support$(y) \subseteq$ support(x). Then $KL(y||x) \geq -\log \sum_{i \in \mathcal{Y}} x_i$.*

Proof. Proved as part of Theorem 2.6.3 in Cover (1999):

$$\begin{aligned} -\mathrm{KL}(y||x) &= \sum_{i \in \mathcal{Y}} y_i \log \frac{x_i}{y_i} \\ &\leq \log \sum_{i \in \mathcal{Y}} y_i \frac{x_i}{y_i} \quad \text{(Jensen's inequality)} \\ &= \log \sum_{i \in \mathcal{Y}} x_i \end{aligned}$$

□

This lower bound is achieved if $x_i = Zy_i$ for all $i \in \mathcal{Y}$, for the constant $Z = \sum_{i \in \mathcal{Y}} x_i$. In this case,

$$\mathrm{KL}(y||x) = \sum_{i \in \mathcal{Y}} y_i \log \frac{y_i}{x_i} = \sum_{i \in \mathcal{Y}} y_i \log \frac{y_i}{Zy_i} = -\sum_{i \in \mathcal{Y}} y_i \log Z = -\log Z\,.$$

If we choose the support of y subject to the condition $|\mathcal{Y}| = m$, this means that $KL(y||x)$ can be minimised by choosing the m support points of x with the highest weight, since this maximises the value of $\log Z$. Hence, we have that choosing the support of y in this way and setting its weights to be proportional to the weights in x is optimal in terms of KL divergence to x. This forms a "greedy resample" of size m from x.

Hence, for a single resample of m particles within a particle filter:

- The resampled distribution with the lowest KL divergence to the original is the one obtained by a greedy resample.
- If the weights of the particle approximation are proportional to the true posterior mass function, the optimal resample of size m from that approximation is a greedy resample, in terms of KL divergence to the true posterior distribution.

If greedy resampling is used at each iteration of the particle filter, then its weights will always be proportional to the true posterior mass function. This does not guarantee that applying greedy resampling **sequentially** will result in an optimal approximation to the true posterior. However, empirical results on the data considered in our experiments showed that particle filters using greedy resampling were able to find clusterings with higher log-posterior mass than those using stochastic resampling.

In addition, in the context of our split SMC algorithm, it follows that it is sometimes optimal to discard particles that introduce overlap between subproblems rather than to perform a merge by taking a sample from their joint distribution (discussed in Sections 3.4 and 3.5). If we assume that the particle filter's weights are proportional to the true posterior, then any resample has KL divergence greater than or equal to that of a greedy resample of the top m clusterings in their joint distribution. If all of the assignments from a subproblem s would be discarded in this greedy resample, then the support of the greedily resampled set is a subset of that of the particle set where

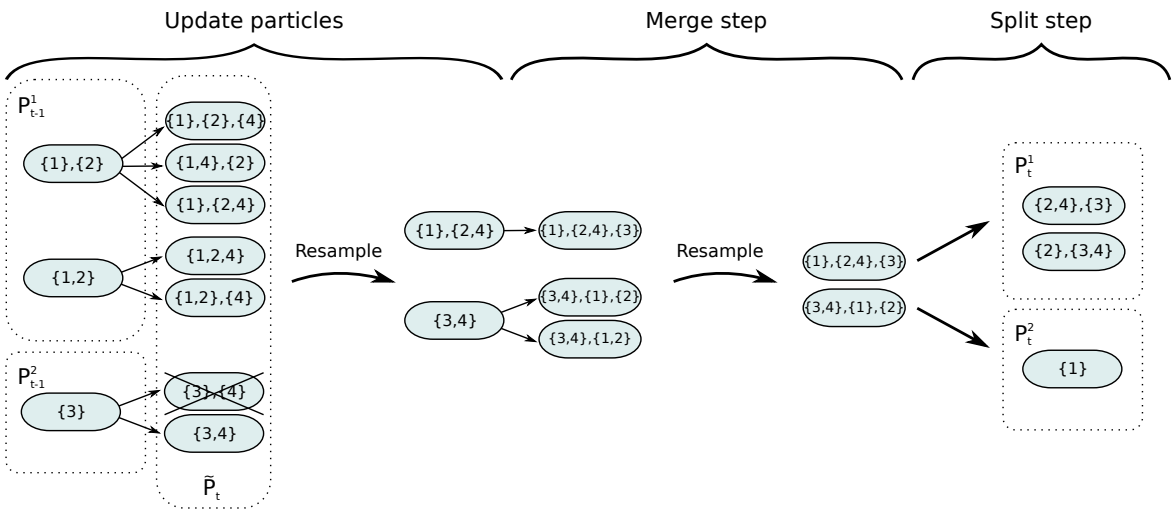

Figure 20: Illustration of a state update in split SMC.

we retain the original subproblem factorisation by discarding only the particles that assign the new observation to clusters in s. Hence the latter particle set has lower KL divergence to the true posterior, and it is optimal not to perform the merge (even if the resampler used in the merge is not greedy).

While checking this condition does not require the construction of the explicit joint distribution in full, it is nonlinear in the number of subproblems under consideration. In practice, faster approximate checks are sufficient for improved accuracy – the rule of thumb we suggest in Section 3.5 discards particles assigning an observations to subproblem s if their combined weight is less than $1/m$, since in this case we expect them to be discarded in the merge.

C SPLIT SMC ALGORITHM DETAILS

Here, we give a detailed description of the full update step in split SMC. An illustration is given in Figure 20, where we suppose that $m = 2$ and we start with a particle set containing clusterings of previous observations $\{1, 2, 3\}$, split into the subproblems $\{1, 2\}$ and $\{3\}$. We refer to this illustration to give a concrete example for each part of the update.

1. **Update particles**
   - Construct the putative particle set $\tilde{\mathcal{P}}_t$, containing a putative particle $\tilde{p}_t(s, i, c)$ for each $1 \leq s \leq S$, $1 \leq i \leq |\mathcal{P}^{(s)}_{t-1}|$, and $c \in p_{t-1}(s, i) \cup \{\varnothing\}$
   - Compute putative weights $\tilde{w}_t(s, i, c) = w^{(i)}_{t-1}\, p\big(x_t \text{ belongs to } c \mid p_{t-1}(s, i), x_{1:(t-1)}\big)$
   - Discard duplicate singleton assignments: for $c = \varnothing$ keep only the $\tilde{p}_t(s, i, c)$ for the subproblem containing the highest value of $p\big(x_t \text{ belongs to } c \mid p_{t-1}(s, i), x_{1:(t-1)}\big)$
   - Resample the m particles with the highest value of $\tilde{w}_t(s, i, c)$ from $\tilde{\mathcal{P}}_t$.

   Having observed new datapoint 4, a putative particle is constructed for each possible cluster assignment. Since the particle set represents a Cartesian product, the putative particle $\{\{3\}, \{4\}\}$ on the second subproblem implicitly represents the particles $\{\{3\}, \{4\}, \{1\}, \{2\}\}$ and $\{\{3\}, \{4\}, \{1, 2\}\}$, which are also contained in the expanded particle set from $\mathcal{P}^1$. The choice of which set of singletons is retained is arbitrary, so we choose the set on the subproblem with the most likely cluster assignment for x_t to reduce the likelihood of needing to perform a merge later. $m = 2$ putative particles are retained after resampling.

2. **Merge**

   If the resampled particles all have the same value of s, then we set $\mathcal{P}^s_t = \tilde{\mathcal{P}}_t$. Otherwise:

- Discard particles with a value of s such that $\sum_{i,c} \tilde{w}_t(s,i,c) \leq 1/m$
- If only one value of s remains, pass to split step
- If at most two values of s have $|P^s_{t-1}| > 1$, compute the full joint distribution over the $\tilde{P}^s_t$. This is done by combining each $\tilde{p}_t(s,i,c)$ with every possible particle configuration p' drawn from the Cartesian product $\bigotimes_{j \in \mathcal{S} \setminus \{s\}} \mathcal{P}^j_{t-1}$. Then we have for each combination and weight

$$\begin{aligned} p_{\text{joint}} &= \tilde{p}_t(s,i,c) \ \cup \ p', \\ w_{\text{joint}} &= \tilde{w}_t(s,i,c) \ \times \prod_{j \in \mathcal{S} \setminus \{s\}} w^{(j,\cdot)}_{t-1} \, . \end{aligned}$$

  Select the m top-weighted clusterings in the above joint distribution.
- Else, perform a multinomial merge: (i) sample m assignments of x_t (with replacement) from the marginal distribution over assignments, then (ii) for each sampled assignment, independently draw one particle from each of the other involved subproblems, and finally (iii) merge any duplicate particles.
- Finally, delete the $\mathcal{P}^s_{t-1}$ from the particle representation, replacing them with a new subproblem consisting of the selected particles.

The resample step selected one particle from each subproblem, so a merge occurs. In our implementation, multinomial resampling is only used in merges of more than two subproblems, where at least 3 of the subproblems have more than one particle, to ensure that the computational cost of the merge step is at most quadratic in m. In practice, this occurs extremely rarely.
Here, only two subproblems are involved, so we form the joint distribution over clusterings of $\{1,2,3,4\}$ by expanding each particle as a Cartesian product with the old generation of particles on the other subproblem. Then, a resample is carried out to bring the number of particles on the merged subproblem down to $m = 2$.

3. Split

- For the subproblem $\mathcal{P}^s_t$ to which x_t was assigned, construct a graph whose vertices are observations and whose edges connect any two points that co-occur in a cluster on at least one particle
- Decompose this graph into its connected components, E_k
- If there is more than one, delete the subproblem and create a new subproblem for each component as

$$\begin{aligned} \mathcal{P}_k \ &= \ \{\, p \cap C_k \mid p \in \mathcal{P}^s_t \}, \\ w^{(k,i)} &= \sum_{j=1}^{|\mathcal{P}_t|} w^{(j)}_t \, \mathbb{1}[\, p^{(k,i)}_t \subseteq p^{(j)}_t \,] \, , \end{aligned}$$

  where C_k is the set of clusters involving points in E_k,

$$C_k \ = \ \Big\{ c \in \bigcup_{p \in \mathcal{P}_t} p : c \subseteq E_k \Big\},$$

Observation 1 does not appear in a cluster with observations 2, 3, or 4 on any particle, so we perform a split assigning it to a separate subproblem.

C.1 Surrogate Proposal

When using a surrogate model as a proposal distribution, m particles with non-empty cluster assignments for the new observation are selected by greedy resampling from the putative particle set, with weights determined by using the surrogate likelihood in the weight update (1). The putative particles where the new observation is assigned to a singleton cluster are also retained. After this proposal step, all particles are re-weighted according to the model likelihood and a further resample down to a total of m particles is carried out. We handle singleton assignments in this way because the value of the Dirichlet process concentration α is typically tuned according to the model likelihood and hence may be poorly calibrated to the surrogate likelihood. Keeping the singleton

assignment particles until the model resampling step prevents them from being discarded prematurely by the surrogate model, ensuring that assignments to non-empty clusters are only weighted highly in the final particle set if they are more likely than the singleton assignment.

If m is small or there is considerable mismatch between the surrogate model and the target model, it can be helpful to propose more than m particles from the surrogate distribution. The size of the proposal step can be chosen according to model evaluation budget, since the number of model evaluations required is at most $m' + 1$, where m' is the number of non-singleton particles selected in the proposal step. We investigate the impact of varying m' in Section A.3. We set $m' = m$ in the rest of our experiments, so that the number of model evaluations scales with the size of the particle set. For the greedy algorithm, m' is set according to evaluation budget since in this case m is fixed at 1. We use $m' = 100$ for the text data, and do not use a surrogate model in greedy clustering of the circles data.

D EXPERIMENTAL DETAILS

The source code for our experiments is available on GitHub at https://github.com/microsoft/smc-clustering. Algorithms were implemented in Python 3 with JAX (Bradbury et al., 2018), SciPy (Virtanen et al., 2020) and NumPy (Harris et al., 2020). The neural model for the circles data was implemented in Flax (Heek et al., 2023) and the neural language model was implemented in PyTorch (Paszke et al., 2019). In all of our algorithm implementations, likelihood evaluations are cached to avoid additional computational cost from re-computing likelihoods. The two REBEL experiments were run on a Linux virtual machine with Ubuntu 24.04.2 LTS on GPU, an A100 with 80Gb of DRAM. All other experiments were run on a cluster with the 22.04.5 LTS "Jammy Jellyfish" 64 bit operating system on CPU (Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz) with a memory limit of 10Gb allocated per replication.

D.1 Offline Algorithms

Offline methods were initialised with the fully disconnected clustering, where every observation is in a singleton cluster. For MCMC samplers, other initialisations are possible, such as the fully-connected clustering that places all observations in a single cluster, and can be helpful for mixing in some applications. The disconnected initialisation is more practical for the clustering problems we consider for two reasons: they have a large underlying number of clusters, and neural likelihood models have an evaluation cost that increases with cluster size.

MCMC We use a Gibbs sampler (Neal, 2000) where at each iteration a Gibbs update is carried out on each datapoint's assignment, in a random order. This involves sampling a new cluster assignment from the marginal distribution defined in (1). For the problems with neural likelihoods, we also considered a Metropolis-within-Gibbs variant. In this sampler, the new assignment is proposed using the surrogate likelihood and is then accepted or rejected according to a Metropolis-Hastings step. This process substantially reduces model evaluations relative to ordinary Gibbs sampling, but can have slower mixing in terms of the number of iterations required for convergence. We compared the two approaches in the circles experiment (see Section A.1), and found that Metropolis-within-Gibbs converged in a much shorter time and obtained slightly superior results to ordinary Gibbs, which did not appear to converge fully in the given runtime.

Agglomerative Clustering In Table 1, we presented results for a full-batch agglomerative algorithm that computes model likelihoods for each possible merged cluster at each iteration. This method has an extremely high runtime due to the number of model evaluations required, and it is possible to formulate a faster, approximate agglomerative algorithm by instead sampling batches of cluster merges to evaluate at each iteration. We present results for a range of batch sizes in Section A.1. They achieved comparable but typically inferior results at lower runtimes than full-batch agglomerative clustering, but were slower than the online algorithms.

D.2 Experiments

Circles 15 cluster centres were drawn from a 2D uniform distribution on $[-5, 5] \times [-5, 5]$, with datapoints drawn uniformly from the circumference of a circle of radius 0.6. The number of points in each cluster was drawn from a uniform categorical distribution on $\{10, \ldots, 30\}$, resulting in a dataset of size 306.

The variational diffusion model (Kingma et al., 2021) was trained on a simulated dataset of 100 000 clusters for 150 epochs using the Adam optimiser. The score function model was a transformer with self attention applied across the sequence of observations within each cluster in order to produce a model for collections of observations (Vaswani et al., 2017). Since we do not use positional encoding, the resulting likelihood model is invariant to permutation of its inputs, fulfilling the criteria we assume for cluster likelihoods in Section 2.1. The architecture used was a residual neural network with 6 residual blocks, each consisting of a self-attention layer, a dense layer of dimension 8, and a dense layer of dimension 2. GELU (Gaussian Error Linear Unit) activation and RMSNorm (Zhang and Sennrich, 2019) were used, with dropout applied during training. A linear noise schedule was used. Cluster likelihoods were estimated by numerical integration of the probability flow ODE. The full implementation of this model can be found in the source code for this paper.

The prior parameters in the surrogate model were tuned by numerically optimising its model evidence on a simulated dataset of 1000 clusters. The Dirichlet process concentration was set to $\alpha = 1$ for all of the clustering algorithms.

Results are reported over 30 replications of the clustering algorithms. SMC samplers were run for a range of particle set sizes with runtimes up to 1000s. MCMC samplers were given a maximum of 10 000s, stopping early if no change was made to the estimated MAP clustering for 500 sweeps. The agglomerative algorithms were run until all potential merges were exhausted, or until 100 iterations passed with no changes to the clustering. Note that the model log-likelihood evaluations are noisy due to use of the Skilling-Hutchinson trace estimator when computing cluster likelihoods. This leads to variability in the results of otherwise deterministic algorithms. Full results for this dataset are plotted in Figure 5.

Gaussian mixture Cluster sizes for 100 clusters and 700 datapoints were generated from a Dirichlet process with concentration $\alpha = 20$ using the stick-breaking construction. Some of the resulting clusters were empty, so the number of clusters present in the generated dataset was 80. The cluster data were generated from Gaussian distributions with independent components. In order to create a problem with multiple close groups of clusters, means were generated in a hierarchical fashion. First, cluster variances were drawn from an inverse Gamma distribution,

$$\sigma_i^{-2} \sim \Gamma(\alpha, \beta)\,.$$

Then 16 group centres were drawn from a Normal distribution,

$$\mu_g \sim N(\mu, \lambda^{-1})\,.$$

Each cluster mean μ_i was set by selecting one of the μ_g uniformly at random and applying an additional Normal perturbation,

$$\mu_i \sim \mu_g + N\left(0, \frac{\sigma_i^2}{125\lambda}\right)\,.$$

The parameters used were $\alpha = 2$, $\beta = 0.5$, $\mu = 0$, and $\lambda = 0.0002$. In clustering, cluster locations and variances were modelled with a Normal-inverse-Gamma distribution with these parameters.

The MCMC sampler was given a maximum of 10 000s, stopping early if no change was made to the estimated MAP clustering for 500 sweeps. The agglomerative algorithms were run until all potential merges were exhausted, or until 100 iterations passed with no changes to the clustering. These results are plotted in Figure 6.

D.2.1 Text Data

Neural model The cluster likelihood model scores a set of fragments by canonically serializing them into JSON and summing grammar-constrained token log-probabilities under a decoder-only Transformer (≈8.8M parameters). We use a hybrid tokenizer with a structural symbol dictionary and BPE pieces applied inside quoted string literals (Sennrich et al., 2016). The model is GPT-style (Vaswani et al., 2017): d_model=256, 10 layers, 8 heads (32-dim), SwiGLU (Shazeer, 2020) feed-forward (d_ff=768), RMSNorm pre-norm (Zhang and Sennrich, 2019), dropout 0.05, RoPE positional encoding (Su et al., 2024), and tied embeddings. A finite-state grammar automaton builds an allowed-token mask at each step, so the sequence log-likelihood is the sum of log-probabilities over valid transitions only (Koo et al., 2024). Cluster (set) likelihoods are the products of these

constrained conditional probabilities over the canonical serialization. Training used AdamW (lr 3e-4, weight decay 0, 1500 warmup steps, cosine decay with 0.1 floor) for 10 epochs on 1.6M training examples from the REBEL train split.

Model calibration While α can be tuned by empirical Bayes methods (Escobar and West, 1995; McAuliffe et al., 2006), this process assumes that the model likelihood is both normalised and well-calibrated, since α is adjusted to make clusterings with high likelihood have high prior probability. We found that for neural models, and neural language models in particular, the resulting values for α did not necessarily give good accuracy. Since rescaling α is equivalent to multiplying each cluster likelihood by a constant in the clustering posterior distribution, tuning α can be seen as equivalent to tuning the normalising constant of the likelihood. We used a labelled validation clustering dataset from REBEL to find a likelihood rescaling that maximised the log-likelihood of the ground truth clustering. The chosen rescaling was equivalent to setting $\alpha = 500$.

N-gram surrogate model The surrogate model used for clustering entity fragments was a character-level bigram model on the *name* attribute. A Dirichlet prior was placed on the probabilities $\theta_{i|h} = \mathbb{P}(\text{next character is } i | \text{previous character is } h)$ for each character history h, as in Chien (2015). The prior parameters $\alpha_{i|h}$ of these Dirichlet distributions were chosen to be a rescaling of the posterior parameter of a fitted Dirichlet-Categorical distribution. Observed n-gram counts were taken from a training subset of the REBEL data, and a uniform Dirichlet prior was used. This is equivalent to setting $\alpha_{i|h} = 1 + \text{count}((h, i))$, the observed count of n-gram (h, i) with plus-one smoothing. A constant multiplicative rescaling factor $c \in (0, 1]$ was used on each $\alpha_{i|h}$ to ensure that the prior distribution in the DPMM was not too concentrated on the fitted n-gram frequencies, allowing for more variation in these frequencies between clusters. This was tuned by numerically optimising the model evidence of the true clustering on the labelled training data.

REBEL Our experiments used two datasets (REBEL-50 and REBEL-200) derived from the REBEL dataset (Huguet Cabot and Navigli, 2021), which is released under a CC BY-NC-SA 4.0 license. This dataset potentially contains personal/sensitive information about people, as its contents are taken directly from text publicly available on Wikipedia. We construct our entity clustering dataset by starting from the full set of REBEL "fragments" (extractions from short text spans) and downsampling to a fixed target size while preserving realistic ambiguity. A uniform fragment sample would yield mostly singleton or trivially separable clusters, and picking random entities would still tend to give clusters whose members differ too much to be challenging. Instead, we define "confusing entities" as distinct entities that share at least one surface form (a matching mention string) with a seed entity, making them plausible collision candidates. We then iteratively build the dataset by selecting 2-5 confusing entities at a time and collecting each entity's fragments (capped at 50) until the total number of unique entities reaches the target. For splitting, each group (a seed entity plus its confusing entities and all of their fragments) is assigned wholesale to a single train, dev, or test split.

TweetNERD We construct a clustering dataset from the TweetNERD-Academic dataset of Mishra et al. (2022), a benchmark dataset in named entity recognition and disambiguation. It is released under a CC BY 4.0 license. TweetNERD-Academic consists of free-text entity mentions extracted from tweets, annotated with corresponding entities in the Wikidata knowledge base (Wikimedia Foundation, 2012). We collate the list of name mentions for each entity, and our clustering dataset is made up of the unique name variations for the 200 largest entities, with resulting cluster sizes from 4 to 148, and 1931 observations in total. In clustering, these are treated as entity fragments, each with a single observation of the *name* attribute.