4th Symposium on Advances in Approximate Bayesian Inference, 2021 1–25

Distribution Compression in Near-linear Time

Anonymous Authors
Anonymous Institution

Abstract

In distribution compression, one aims to accurately summarize a probability distribution $\mathbb{P}$ using a small number of representative points. Near-optimal thinning procedures achieve this goal by sampling n points from a Markov chain and identifying $\sqrt{n}$ points with $\widetilde{\mathcal{O}}(1/\sqrt{n})$ discrepancy to $\mathbb{P}$. Unfortunately, these algorithms suffer from quadratic or super-quadratic runtime in the sample size n. To address this deficiency, we introduce Compress++, a simple meta-procedure for speeding up any thinning algorithm while suffering at most a factor of 4 in error. When combined with the quadratic-time kernel halving and kernel thinning algorithms of Dwivedi and Mackey (2021), Compress++ delivers $\sqrt{n}$ points with better-than-Monte-Carlo maximum mean discrepancy in $\mathcal{O}(n \log^3 n)$ time and $\mathcal{O}(\sqrt{n} \log^2 n)$ space. Moreover, Compress++ enjoys the same near-linear runtime given any quadratic-time input and reduces the runtime of super-quadratic algorithms by a square-root factor. In our benchmarks with high-dimensional Monte Carlo samples and Markov chains targeting challenging differential equation posteriors, Compress++ matches or nearly matches the accuracy of its input algorithm in orders of magnitude less time.

1. Introduction

Distribution compression—constructing a concise summary of a probability distribution—is at the heart of many learning and inference tasks. For example, in Monte Carlo integration and Bayesian inference, n representative points are sampled i.i.d. or from a Markov chain to approximate expectations and quantify uncertainty under an intractable (posterior) distribution (Robert and Casella, 1999). However, these standard sampling strategies represent a bottleneck in computationally-demanding settings due to their slow root-n Monte Carlo error rate. For instance, the Monte Carlo estimate $\mathbb{P}_{\text{in}} f \triangleq \frac{1}{n}\sum_{i=1}^{n} f(x_i)$ of an unknown expectation $\mathbb{P}f \triangleq \mathbb{E}_{X\sim\mathbb{P}}[f(X)]$ based on n i.i.d. points has $\Theta(n^{-\frac{1}{2}})$ integration error $|\mathbb{P}f - \mathbb{P}_{\text{in}} f|$, requiring $n = 10000$ points for 1% relative error and $n = 10^6$ points for 0.1% error. Such bloated sample representations preclude downstream applications with critically expensive function evaluations like computational cardiology, where a 1000-CPU-hour tissue or organ simulation is required for each sample point (Niederer et al., 2011; Augustin et al., 2016; Strocchi et al., 2020).

To restore the feasibility of such critically expensive tasks, it is common to thin down the initial sequence of points to a produce a much smaller coreset. The standard thinning approach, select every t-th sample point (Owen, 2017), is simple to implement but often leads to an substantial increase in error: e.g., standard thinning n points from a fast-mixing Markov chain yields $\Omega(n^{-\frac{1}{4}})$ error when $n^{\frac{1}{2}}$ points are returned. Recently, Dwivedi and Mackey (2021b,a) introduced a more effective alternative, *kernel thinning*, that provides near optimal $\widetilde{\mathcal{O}}_d(n^{-\frac{1}{2}})$ error when compressing n points in $\mathbb{R}^d$ down to size $n^{\frac{1}{2}}$. While practical for moderate sample sizes, the runtime of this algorithm scales quadratically with the input size n, making its execution prohibitive for very large input sizes. Our goal is to significantly improve the runtime of such compression algorithms while providing comparable error guarantees.

Given a sequence $\mathcal{S}_{\text{in}}$ of n input points summarizing a target distribution $\mathbb{P}$, our aim is to identify a high quality coreset $\mathcal{S}_{\text{out}}$ of size $\sqrt{n}$ in time nearly linear in n. We measure coreset quality via its integration error $|\mathbb{P}f - \mathbb{P}_{\mathcal{S}_{\text{out}}}f| \triangleq |\mathbb{P}f - \frac{1}{|\mathcal{S}_{\text{out}}|}\sum_{x\in\mathcal{S}_{\text{out}}} f(x)|$ for functions f in the reproducing kernel Hilbert space (RKHS) $\mathcal{H}_{\mathbf{k}}$ induced by a given kernel $\mathbf{k}$ (Berlinet and Thomas-Agnan, 2011). We consider both single function error and kernel *maximum mean discrepancy* (MMD, Gretton et al., 2012), the worst-case integration error over the unit RKHS norm ball:

$$\text{MMD}_{\mathbf{k}}(\mathbb{P}, \mathbb{P}_{\mathcal{S}_{\text{out}}}) \triangleq \sup_{\|f\|_{\mathbf{k}}\leq 1} |\mathbb{P}f - \mathbb{P}_{\mathcal{S}_{\text{out}}}f|. \tag{1}$$

We introduce a new simple meta procedure—COMPRESS++—that significantly speeds up a generic thinning algorithm while simultaneously inheriting the error guarantees of its input up to a factor of four. A direct application of COMPRESS++ to kernel thinning improves its quadratic $\Theta(n^2)$ runtime to near linear $\mathcal{O}(n \log^3 n)$ time while preserving its error guarantees. Since the $\widetilde{\mathcal{O}}_d(n^{-\frac{1}{2}})$ KT MMD guarantees of Dwivedi and Mackey (2021b) match the $\Omega(n^{-\frac{1}{2}})$ minimax lower bounds of Tolstikhin et al. (2017); Phillips and Tai (2020) up to factors of $\sqrt{\log(n)}$ and constants depending on d, KT-COMPRESS++ also provides near-optimal MMD compression for a wide range of kernels and distributions $\mathbb{P}$. Moreover, the practical gains from applying COMPRESS++ are substantial: KT thins $65,000$ points in 10 dimensions in 20m, while KT-COMPRESS++ needs only 1.5m; KT takes more than a day to thin $250,000$ points in 100 dimensions, while KT-COMPRESS++ takes less than an hour (a $32\times$ speed-up). For larger n, the speed-ups are even greater due to the order $\frac{n}{\log^3 n}$ reduction in runtime.

COMPRESS++ can also be directly combined with any thinning algorithm, even those that have suboptimal guarantees but often perform well in practice, like kernel herding (Chen et al., 2010), MMD-critic (Kim et al., 2016), and Stein thinning (Riabiz et al., 2020a), all of which run in $\Omega(n^2)$ time.

We let $\mathbb{P}_{\mathcal{S}}$ denote the empirical distribution of $\mathcal{S}$. For the output coreset $\mathcal{S}_{\text{ALG}}$ of the algorithm ALG with the input coreset $\mathcal{S}_{\text{in}}$, we use the simpler notation $\mathbb{P}_{\text{ALG}} \triangleq \mathbb{P}_{\mathcal{S}_{\text{ALG}}}$, along with $\mathbb{P}_{\text{in}} \triangleq \mathbb{P}_{\mathcal{S}_{\text{in}}}$. We extend our MMD definition to point sequences $(\mathcal{S}_1, \mathcal{S}_2)$ via $\text{MMD}_{\mathbf{k}}(\mathcal{S}_1, \mathcal{S}_2) \triangleq \text{MMD}_{\mathbf{k}}(\mathbb{P}_{\mathcal{S}_1}, \mathbb{P}_{\mathcal{S}_2})$ and $\text{MMD}_{\mathbf{k}}(\mathbb{P}, \mathcal{S}_1) \triangleq \text{MMD}_{\mathbf{k}}(\mathbb{P}, \mathbb{P}_{\mathcal{S}_1})$.

2. Thinning and Halving Algorithms

We begin by defining the thinning and halving algorithms that our meta-procedures take as input.

Definition 1 (Thinning and halving algorithms) *A* thinning algorithm ALG *takes as input a point sequence $\mathcal{S}_{\text{in}}$ of length n and returns a (possibly random) point sequence $\mathcal{S}_{\text{ALG}}$ of length n_{out}. We say* ALG *is* α_n-thinning *if $n_{\text{out}} = \lfloor n/\alpha_n \rfloor$ and* root-thinning *if $\alpha_n = \sqrt{n}$. Moreover, we call* ALG *a* halving algorithm *if $\mathcal{S}_{\text{ALG}}$ always contains exactly $\lfloor \frac{n}{2} \rfloor$ of the input points.*

Many algorithms also offer high-probability bounds on the kernel MMD (1), the worst-case integration error across the unit ball of the RKHS. We again capture these bounds abstractly using the following definition of a $\mathbf{k}$-sub-Gaussian thinning algorithm.

Definition 2 ($\mathbf{k}$-sub-Gaussian thinning algorithm) *For a kernel $\mathbf{k}$, we call a thinning algorithm* ALG $\mathbf{k}$-sub-Gaussian *with parameter v and shift a and write* $\text{ALG} \in \mathcal{G}_{\mathbf{k}}(v, a)$ *if*

$$\mathbb{P}[\text{MMD}_{\mathbf{k}}(\mathcal{S}_{\text{in}}, \mathcal{S}_{\text{ALG}}) \geq a_n + v_n\sqrt{t} \,\big|\, \mathcal{S}_{\text{in}}] \leq e^{-t} \quad \textit{for all} \quad t \geq 0. \tag{2}$$

We also call $\varepsilon_{\mathbf{k},\text{ALG}}(n) \triangleq \max(v_n, a_n)$ the $\mathbf{k}$-sub-Gaussian error *of* ALG.

3. COMPRESS

The core subroutine of COMPRESS++ is a new meta-procedure called COMPRESS that, given a halving algorithm HALVE, an oversampling parameter $\mathfrak{g}$, and n input points, outputs a thinned coreset of size $2^{\mathfrak{g}}\sqrt{n}$. The COMPRESS algorithm (Alg. 1) is very simple to implement: first, divide the input points into four subsequences of size $\frac{n}{4}$ (in any manner the user chooses); second, recursively call COMPRESS on each subsequence to produce four coresets of size $2^{\mathfrak{g}-1}\sqrt{n}$; finally, call HALVE on the concatenation of those coresets to produce the final output of size $2^{\mathfrak{g}}\sqrt{n}$. As we show in App. K, COMPRESS can also be implemented in a streaming fashion to consume at most $\mathcal{O}(4^{\mathfrak{g}}\sqrt{n})$ memory when processing n input points.

Algorithm 1: COMPRESS

Input: halving algorithm HALVE, oversampling parameter $\mathfrak{g}$, point sequence $\mathcal{S}_{\text{in}}$ of size n
if $n = 4^{\mathfrak{g}}$ **then return** $\mathcal{S}_{\text{in}}$
else
  Partition $\mathcal{S}_{\text{in}}$ into four arbitrary subsequences $\{\mathcal{S}_i\}_{i=1}^4$ each of size $n/4$
  for $i = 1, 2, 3, 4$ **do**
    $\widetilde{\mathcal{S}}_i \leftarrow$ COMPRESS$(\mathcal{S}_i, \text{HALVE}, \mathfrak{g})$ // return coresets of size $2^{\mathfrak{g}} \cdot \sqrt{\frac{n}{4}}$
  end
  $\widetilde{\mathcal{S}} \leftarrow$ CONCATENATE$(\widetilde{\mathcal{S}}_1, \widetilde{\mathcal{S}}_2, \widetilde{\mathcal{S}}_3, \widetilde{\mathcal{S}}_4)$ // coreset of size $2 \cdot 2^{\mathfrak{g}} \cdot \sqrt{n}$
  return HALVE$(\widetilde{\mathcal{S}})$ // coreset of size $2^{\mathfrak{g}}\sqrt{n}$
end

Our first result relates the runtime of COMPRESS to the runtime and error of HALVE. We measure runtime by the number of dominant operations performed by HALVE (e.g., the number of kernel evaluations performed by kernel thinning).

Theorem 3 (Runtime of COMPRESS) *If* HALVE *has runtime* $r_{\text{H}}(n)$ *for inputs of size* n*, then* COMPRESS *has runtime*

$$r_{\text{C}}(n) = \textstyle\sum_{i=0}^{\beta_n} 4^i \cdot r_{\text{H}}(\ell_n 2^{-i}), \tag{3}$$

where $\ell_n \triangleq 2^{\mathfrak{g}+1}\sqrt{n}$ *(twice the output size of* COMPRESS*), and* $\beta_n \triangleq \log_2(\frac{n}{\ell_n}) = \log_4 n - \mathfrak{g} - 1$.

As we prove in App. C, the runtime guarantee (3) is immediate once we unroll the COMPRESS recursion and identify that COMPRESS makes 4^i calls to HALVE with input size $\ell_n 2^{-i}$. Let us now unpack the most important implications of Thm. 3.

Remark 4 (Near-linear runtime and quadratic speed-ups for COMPRESS) Thm. 3 implies that a quadratic-time HALVE with $r_{\text{H}}(n) = n^2$ yields a near-linear time COMPRESS with $r_{\text{C}}(n) \leq 4^{\mathfrak{g}+1} n(\log_4(n) - \mathfrak{g})$. If HALVE instead has super-quadratic runtime $r_{\text{H}}(n) = n^\tau$, COMPRESS enjoys a quadratic speed-up: $r_{\text{C}}(n) \leq c'_\tau n^{\tau/2}$ for $c'_\tau \triangleq \frac{2^{\tau(\mathfrak{g}+2)}}{2^\tau - 4}$. More generally, whenever HALVE has superlinear runtime $r_{\text{H}}(n) = n^\tau \rho(n)$ for some $\tau \geq 1$ and non-decreasing ρ, COMPRESS satisfies

$$r_{\text{C}}(n) \leq \begin{cases} c_\tau \cdot n\,(\log_4(n) - \mathfrak{g})\,\rho(\ell_n) & \text{for } \tau \leq 2 \\ c'_\tau \cdot n^{\tau/2}\,\rho(\ell_n) & \text{for } \tau > 2 \end{cases} \quad \text{where} \quad c_\tau \triangleq 4^{(\tau-1)(\mathfrak{g}+1)}.$$

Next, we bound the MMD error of COMPRESS in terms of the MMD error of HALVE. Recall that $\text{MMD}_{\mathbf{k}}$ (1) represents the worst-case integration error across the unit ball of the RKHS of $\mathbf{k}$. The error guarantee is subtle: here, COMPRESS benefits significantly from random cancellations among the *conditionally independent* and *mean-zero* HALVE errors. Without these properties, the errors from each HALVE call could compound without cancellation leading to a significant degradation in COMPRESS quality. The proof, based on the concentration of subexponential matrix martingales, is provided in App. D.

Theorem 5 (MMD guarantees for COMPRESS) *Suppose* HALVE $\in \mathcal{G}_{\mathbf{k}}(a, v)$ *for* n a_n *and* n v_n *non-decreasing and* $\mathbb{E}\left[\mathbb{P}_{\text{HALVE}}\mathbf{k} \mid \mathcal{S}_{\text{in}}\right] = \mathbb{P}_{\text{in}}\mathbf{k}$. *Then* COMPRESS $\in \mathcal{G}_{\mathbf{k}}(\widetilde{a}, \widetilde{v})$ *with*

$$\widetilde{v}_n \triangleq \widetilde{c}_n\sqrt{2(\log_4 n - \mathfrak{g})}, \quad \textit{and} \quad \widetilde{a}_n \triangleq \widetilde{c}_n \log(n+1) + \widetilde{v}_n\sqrt{\log(n+1)}, \tag{4}$$

where $\widetilde{c}_n \triangleq 2(a_{\ell_n} + v_{\ell_n})$ *and* $\ell_n = 2^{\mathfrak{g}+1}\sqrt{n}$ *as in Thm. 3.*

Remark 6 (COMPRESS inflates MMD guarantee by at most $9\log(n+1)$) Thm. 5 implies that the $\mathbf{k}$-sub-Gaussian error of COMPRESS is always at most $9\log(n+1)$ times that of HALVE with input size $\ell_n = 2^{\mathfrak{g}+1}\sqrt{n}$ since

$$\varepsilon_{\mathbf{k},\text{COMPRESS}}(n) \overset{\text{Thm. 2}}{=} \max(\widetilde{a}_n, \widetilde{v}_n) \overset{(4)}{\leq} 9\log(n+1)\max(a_{\ell_n}, v_{\ell_n}) = 9\log(n+1)\cdot\varepsilon_{\mathbf{k},\text{HALVE}}(\ell_n).$$

HALVE applied to an input of size ℓ_n is a particularly strong benchmark, as ℓ_n is twice the output size of COMPRESS, and thinning from n to $\frac{\ell_n}{2}$ points should incur at least as much MMD error as halving from ℓ_n to $\frac{\ell_n}{2}$.

4. COMPRESS++

To offset any excess error due to COMPRESS while maintaining its near-linear runtime, we next introduce COMPRESS++ (Alg. 2), a simple two-stage meta-procedure for faster root-thinning. COMPRESS++ takes as input an oversampling parameter $\mathfrak{g}$, a halving algorithm HALVE, and a $2^{\mathfrak{g}}$-thinning algorithm THIN (see Thm. 1). In our applications, HALVE and THIN are derived from the same base algorithm (e.g., from kernel thinning with different thinning factors), but this is not required. COMPRESS++ first runs the faster but slightly more erroneous COMPRESS(HALVE, $\mathfrak{g}$) algorithm to produce an intermediate coreset of size $2^{\mathfrak{g}}\sqrt{n}$. Next, the slower but more accurate THIN algorithm is run on the greatly compressed intermediate coreset to produce a final output of size $\sqrt{n}$. In the sequel we will demonstrate how to set $\mathfrak{g}$ to offset error inflation due to COMPRESS while maintaining its fast runtime.

Algorithm 2: COMPRESS++

Input: oversampling parameter $\mathfrak{g}$, halving algorithm HALVE, $2^{\mathfrak{g}}$-thinning algorithm THIN, point sequence $\mathcal{S}_{\text{in}}$ of size n

$\mathcal{S}_{\text{C}} \leftarrow$ COMPRESS(HALVE, $\mathfrak{g}, \mathcal{S}_{\text{in}}$) // Coreset of size $2^{\mathfrak{g}}\sqrt{n}$
$\mathcal{S}_{\text{C++}} \leftarrow$ THIN($\mathcal{S}_{\text{C}}$) // Coreset of size $\sqrt{n}$
return $\mathcal{S}_{\text{C++}}$

The following result, proved in App. F, relates the runtime of COMPRESS++ to the runtime of HALVE and THIN.

Theorem 7 (Runtime and integration error of COMPRESS++) *If* HALVE *and* THIN *have runtimes* $r_{\text{H}}(n)$ *and* $r_{\text{T}}(n)$ *respectively for inputs of size* n, *then* COMPRESS++ *has runtime*

$$r_{\text{C++}}(n) = r_{\text{C}}(n) + r_{\text{T}}(\ell_n/2) \quad \textit{where} \quad r_{\text{C}}(n) \overset{(3)}{=} \sum_{i=0}^{\beta_n} 4^i \cdot r_{\text{H}}(\ell_n 2^{-i}), \tag{5}$$

$\ell_n = 2^{\mathfrak{g}+1}\sqrt{n}$, *and* $\beta_n = \log_4 n - \mathfrak{g} - 1$ *as in Thm. 3.*

Remark 8 (Near-linear runtime and near-quadratic speed-ups for COMPRESS++) When HALVE and THIN have quadratic runtimes with $\max(r_{\text{H}}(n), r_{\text{T}}(n)) = n^2$, Thms. 4 and 7 yield that $r_{\text{C++}}(n) \leq 4^{\mathfrak{g}+1}\, n(\log_4(n) - \mathfrak{g}) + 4^{\mathfrak{g}} n$. Hence, COMPRESS++ maintains a near-linear runtime

$$r_{\text{C++}}(n) = \mathcal{O}(n\log_4^{c+1}(n)) \quad \text{whenever} \quad 4^{\mathfrak{g}} = \mathcal{O}(\log_4^c n).$$

If HALVE and THIN instead have super-quadratic runtimes with $\max(r_{\text{H}}(n), r_{\text{T}}(n)) = n^\tau$, then by Thm. 4 we have $r_{\text{C++}}(n) \leq (\frac{4^\tau}{2^\tau - 4} + 1)\, 2^{\mathfrak{g}\tau} n^{\tau/2}$, so that COMPRESS++ provides a near-quadratic speed up $r_{\text{C++}}(n) = \mathcal{O}(n^{\tau/2} \log_4^{c\tau/2}(n))$ whenever $4^{\mathfrak{g}} = \mathcal{O}(\log_4^c n)$.

Next, we bound the MMD error of COMPRESS++ in terms of the MMD error of HALVE and THIN. The proof of the following result can be found in App. G.

Theorem 9 (MMD guarantees for COMPRESS++) *If* THIN $\in \mathcal{G}_{\mathbf{k}}(a', v')$, HALVE $\in \mathcal{G}_{\mathbf{k}}(a, v)$ *for* $n\, a_n$ *and* $n\, v_n$ *non-decreasing, and* $\mathbb{E}\left[\mathbb{P}_{\text{HALVE}}\mathbf{k} \mid \mathcal{S}_{\text{in}}\right] = \mathbb{P}_{\text{in}}\mathbf{k}$, *then* COMPRESS++ $\in \mathcal{G}_{\mathbf{k}}(\widehat{a}, \widehat{v})$ *with*

$$\widehat{v}_n \triangleq \widetilde{v}_n + v'_{\ell_n/2}, \quad \textit{and} \quad \widehat{a}_n \triangleq \widetilde{a}_n + a'_{\ell_n/2} + \widehat{v}_n \sqrt{\log 2}$$

for $\widetilde{v}_n$ *and* $\widetilde{a}_n$ *defined in Thm. 5 and* $\ell_n = 2^{\mathfrak{g}+1}\sqrt{n}$ *as in Thm. 3.*

Remark 10 (COMPRESS++ inflates MMD guarantee by at most 4) Thm. 9 implies that the COMPRESS++ $\mathbf{k}$-sub-Gaussian error $\varepsilon_{\mathbf{k},\text{COMPRESS++}}(n) = \max(\widehat{a}_n, \widehat{v}_n)$ satisfies

$$\begin{aligned}\varepsilon_{\mathbf{k},\text{COMPRESS++}}(n) &\leq (9\log(n+1)\, \varepsilon_{\mathbf{k},\text{HALVE}}(\ell_n) + \varepsilon_{\mathbf{k},\text{THIN}}(\tfrac{\ell_n}{2}))\,(1 + \sqrt{\log 2}) \\ &\leq \varepsilon_{\mathbf{k},\text{THIN}}(\tfrac{\ell_n}{2})(\tfrac{9\log(n+1)}{2^{\mathfrak{g}}} \tfrac{\widetilde{\zeta}_{\text{H}}(\ell_n)}{\widetilde{\zeta}_{\text{T}}(\frac{\ell_n}{2})} + 1)(1 + \sqrt{\log 2}),\end{aligned}$$

where we have introduced the rescaled quantities $\widetilde{\zeta}_{\text{H}}(\ell_n) \triangleq \frac{\ell_n}{2}\, \varepsilon_{\mathbf{k},\text{HALVE}}(\ell_n)$ and $\widetilde{\zeta}_{\text{T}}(\frac{\ell_n}{2}) \triangleq \sqrt{n}\, \varepsilon_{\mathbf{k},\text{THIN}}(\frac{\ell_n}{2})$. Therefore, COMPRESS++ satisfies

$$\varepsilon_{\mathbf{k},\text{COMPRESS++}}(n) \leq 4\, \varepsilon_{\mathbf{k},\text{THIN}}(\tfrac{\ell_n}{2}) \quad \text{whenever} \quad \mathfrak{g} \geq \log_2 \log(n+1) + \log_2(8 \tfrac{\widetilde{\zeta}_{\text{H}}(\ell_n)}{\widetilde{\zeta}_{\text{T}}(\frac{\ell_n}{2})}). \tag{6}$$

In other words, relative to a strong baseline of thinning from $\frac{\ell_n}{2}$ to $\sqrt{n}$ points, COMPRESS++ inflates $\mathbf{k}$-sub-Gaussian error by at most a factor of 4 whenever $\mathfrak{g}$ satisfies (6). For example, when the ratio $\widetilde{\zeta}_{\text{H}}(\ell_n)/\widetilde{\zeta}_{\text{T}}(\frac{\ell_n}{2})$ is bounded by C, it suffices to choose $\mathfrak{g} = \lceil \log_2 \log(n+1) + \log_2(8C) \rceil$.

5. Experiments

Supplementary experimental details and results can be found in App. J.

Each experiment compares a high-accuracy, quadratic time thinning algorithm—either target kernel thinning (Dwivedi and Mackey, 2021a) —with our near-linear time COMPRESS and COMPRESS++ variants that use the same input algorithm to HALVE and THIN. In each case, we perform root thinning, compressing n input points down to $\sqrt{n}$ points, so that COMPRESS is run with $\mathfrak{g} = 0$. For COMPRESS++, we use $\mathfrak{g} = 4$ throughout to satisfy the small relative error criterion (6) in all experiments. When halving we restrict each input algorithm to return distinct points and symmetrize the output as discussed in App. E.

We first apply COMPRESS++ to the near-optimal KT algorithm to obtain comparable summaries at a fraction of the cost. Figs. 3 and 4 reveal that, in line with our guarantees, KT-COMPRESS++ matches or nearly matches the MMD error of KT in all experiments while also substantially reducing runtime. For example, KT thins $65,000$ points in 10 dimensions in 20m, while KT-COMPRESS++ needs only 1.5m; KT takes more than a day to thin $250,000$ points in 100 dimensions, while KT-COMPRESS++ takes less than an hour (a $32\times$ speed-up).

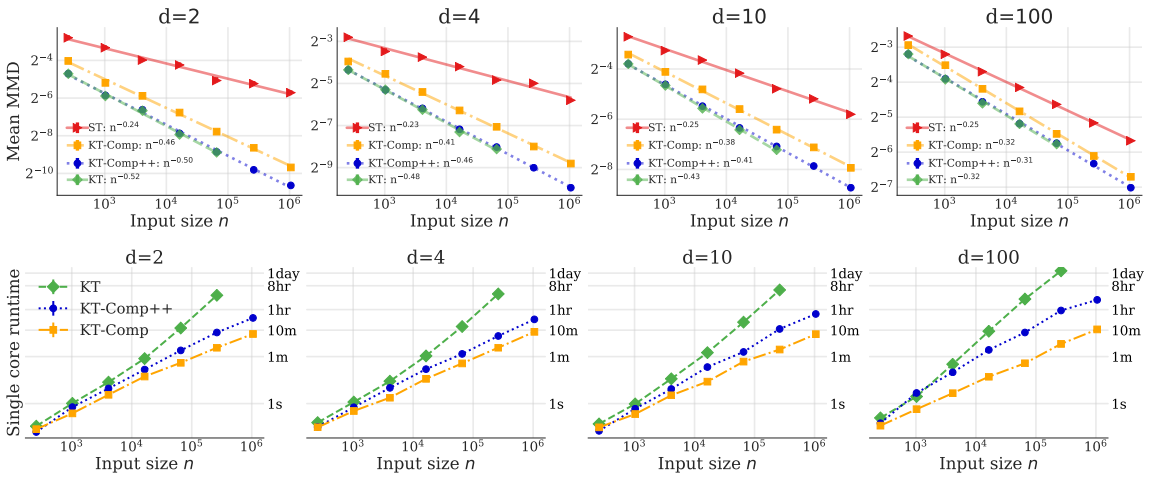

Figure 1: For Gaussian targets $\mathbb{P}$ with $d \in \{2, 4, 10, 100\}$, KT-COMPRESS++ improves upon the MMD of i.i.d. sampling (ST), closely tracks the error of its quadratic-time input algorithm KT, and substantially reduces the runtime. See App. J.2 for more details.

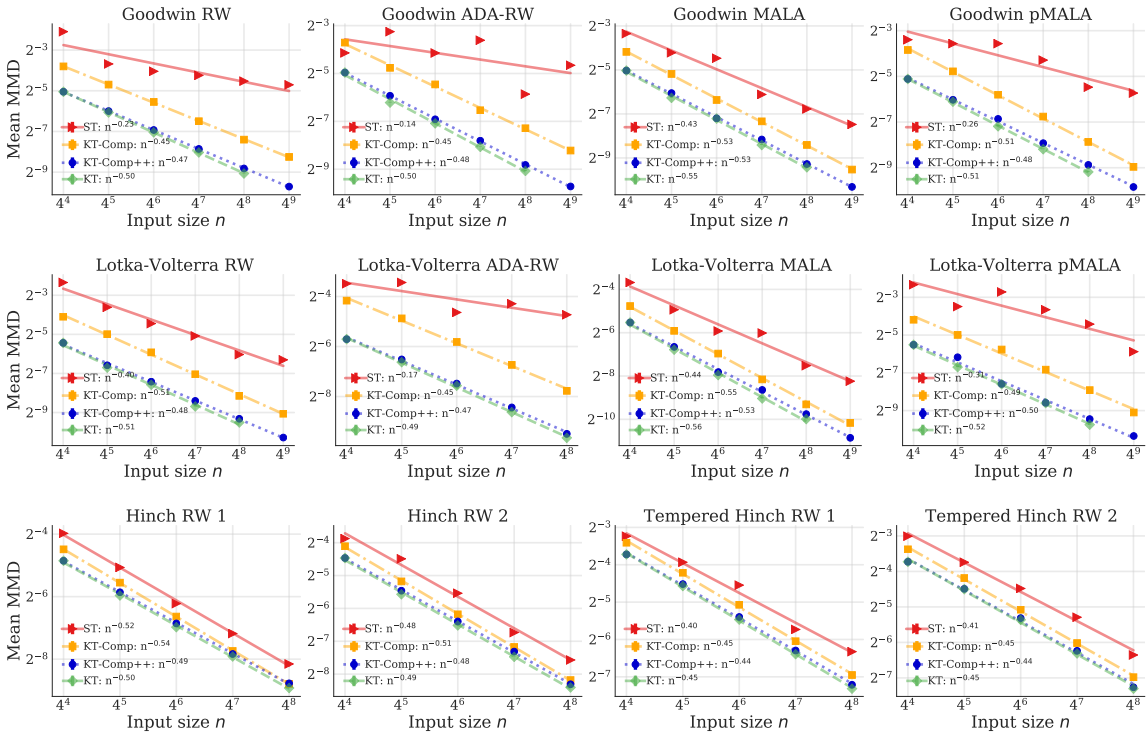

Figure 2: Given MCMC sequences summarizing challenging differential equation posteriors $\mathbb{P}$, KT-COMPRESS++ consistently improves upon the MMD of standard thinning (ST) and matches or nearly matches the error of of its quadratic-time input algorithm KT. See App. J.2 for more details.

References

Christoph M Augustin, Aurel Neic, Manfred Liebmann, Anton J Prassl, Steven A Niederer, Gundolf Haase, and Gernot Plank. Anatomically accurate high resolution modeling of human whole heart electrome-

chanics: A strongly scalable algebraic multigrid solver method for nonlinear deformation. *Journal of computational physics*, 305:622–646, 2016.

Alain Berlinet and Christine Thomas-Agnan. *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media, 2011.

S. Boucheron, G. Lugosi, and P. Massart. *Concentration Inequalities: A Nonasymptotic Theory of Independence*. OUP Oxford, 2013. ISBN 9780199535255. URL https://books.google.com/books?id=koNqWRluhP0C.

Yutian Chen, Max Welling, and Alex Smola. Super-samples from kernel herding. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, UAI'10, page 109–116, Arlington, Virginia, USA, 2010. AUAI Press. ISBN 9780974903965.

Raaz Dwivedi and Lester Mackey. Generalized kernel thinning. *arXiv preprint arXiv:2110.01593*, 2021a.

Raaz Dwivedi and Lester Mackey. Kernel thinning. *arXiv preprint arXiv:2105.05842*, 2021b.

Mark Girolami and Ben Calderhead. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, 2011.

Brian C Goodwin. Oscillatory behavior in enzymatic control process. *Advances in Enzyme Regulation*, 3: 318–356, 1965.

Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773, 2012.

Heikki Haario, Eero Saksman, and Johanna Tamminen. Adaptive proposal distribution for random walk Metropolis algorithm. *Computational Statistics*, 14(3):375–395, 1999.

Robert Hinch, JL Greenstein, AJ Tanskanen, L Xu, and RL Winslow. A simplified local control model of calcium-induced calcium release in cardiac ventricular myocytes. *Biophysical journal*, 87(6):3723–3736, 2004.

Been Kim, Rajiv Khanna, and Oluwasanmi O Koyejo. Examples are not enough, learn to criticize! criticism for interpretability. *Advances in neural information processing systems*, 29, 2016.

Alfred James Lotka. *Elements of physical biology*. Williams & Wilkins, 1925.

Steven A Niederer, Lawrence Mitchell, Nicolas Smith, and Gernot Plank. Simulating human cardiac electrophysiology on clinical time-scales. *Frontiers in Physiology*, 2:14, 2011.

Art B Owen. Statistically efficient thinning of a Markov chain sampler. *Journal of Computational and Graphical Statistics*, 26(3):738–744, 2017.

Jeff M Phillips and Wai Ming Tai. Near-optimal coresets of kernel density estimates. *Discrete & Computational Geometry*, 63(4):867–887, 2020.

Marina Riabiz, Wilson Chen, Jon Cockayne, Pawel Swietach, Steven A Niederer, Lester Mackey, and Chris Oates. Optimal thinning of MCMC output. *arXiv preprint arXiv:2005.03952*, 2020a.

Marina Riabiz, Wilson Ye Chen, Jon Cockayne, Pawel Swietach, Steven A. Niederer, Lester Mackey, and Chris J. Oates. Replication Data for: Optimal Thinning of MCMC Output, 2020b. URL https://doi.org/10.7910/DVN/MDKNWM. Accessed on Mar 23, 2021.

Christian P Robert and George Casella. Monte Carlo integration. In *Monte Carlo statistical methods*, pages 71–138. Springer, 1999.

Gareth O Roberts and Richard L Tweedie. Exponential convergence of Langevin distributions and their discrete approximations. *Bernoulli*, 2(4):341–363, 1996.

Marina Strocchi, Matthias AF Gsell, Christoph M Augustin, Orod Razeghi, Caroline H Roney, Anton J Prassl, Edward J Vigmond, Jonathan M Behar, Justin S Gould, Christopher A Rinaldi, Martin J Bishop, Gernot Plank, and Steven A Niederer. Simulating ventricular systolic motion in a four-chamber heart model with spatially varying robin boundary conditions to model the effect of the pericardium. *Journal of Biomechanics*, 101:109645, 2020.

Ilya Tolstikhin, Bharath K Sriperumbudur, and Krikamol Muandet. Minimax estimation of kernel mean embeddings. *The Journal of Machine Learning Research*, 18(1):3002–3048, 2017.

Joel Tropp. Freedman's inequality for matrix martingales. *Electronic Communications in Probability*, 16 (none):262 – 270, 2011. doi: 10.1214/ECP.v16-1624. URL https://doi.org/10.1214/ECP.v16-1624.

Joel A. Tropp. User-friendly tail bounds for sums of random matrices. *Foundations of Computational Mathematics*, 12(4):389–434, 2012. doi: 10.1007/s10208-011-9099-z. URL https://doi.org/10.1007/s10208-011-9099-z.

Vito Volterra. Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. 1926.

Appendix

Appendix A. Additional Definitions and Notation

This section provides additional definitions and notation used throughout the appendices.

We associate with each algorithm ALG and input $\mathcal{S}_{\text{in}}$ the measure difference

$$\phi_{\text{ALG}}(\mathcal{S}_{\text{in}}) \triangleq \mathbb{P}_{\mathcal{S}_{\text{in}}} - \mathbb{P}_{\mathcal{S}_{\text{ALG}}} = \frac{1}{n}\sum_{x\in\mathcal{S}_{\text{in}}}\delta_x - \frac{1}{n_{\text{out}}}\sum_{x\in\mathcal{S}_{\text{ALG}}}\delta_x \tag{7}$$

that characterizes how well the output empirical distribution approximates the input. We will often write ϕ_{ALG} instead of $\phi_{\text{ALG}}(\mathcal{S}_{\text{in}})$ for brevity if $\mathcal{S}_{\text{in}}$ is clear from the context.

We also make use of the following standard definition of a sub-Gaussian random variable (see, e.g., Boucheron et al., 2013, Sec. 2.3).

Definition 11 (Sub-Gaussian random variable) *We say that a random variable G is sub-Gaussian with parameter ν and write $G \in \mathcal{G}(\nu)$ if*

$$\mathbb{E}\big[\exp(\lambda G)\big] \leq \exp\Big(\frac{\lambda^2\nu^2}{2}\Big) \quad \textit{for all} \quad \lambda \in \mathbb{R}.$$

Given Thm. 11, it follows that $\text{ALG} \in \mathcal{G}^f(\nu)$ for a function f as in Thm. 13 if and only if the random variable $\phi_{\text{ALG}}(f) \triangleq \mathbb{P}_{\mathcal{S}_{\text{in}}} f - \mathbb{P}_{\mathcal{S}_{\text{ALG}}} f$ is sub-Gaussian with parameter ν conditional on the input $\mathcal{S}_{\text{in}}$.

In our proofs, it is often more convenient to work with an unnormalized *measure discrepancy*

$$\psi_{\text{ALG}}(\mathcal{S}_{\text{in}}) \triangleq n \cdot \phi_{\text{ALG}}(\mathcal{S}_{\text{in}}) \overset{(7)}{=} \sum_{x \in \mathcal{S}_{\text{in}}} \delta_x - \frac{n}{n_{\text{out}}} \sum_{\mathcal{S}_{\text{ALG}}} \delta_x. \tag{8}$$

By definition (8), we have the following useful equivalence:

$$\psi_{\text{ALG}}(f) \triangleq n \cdot \phi_{\text{ALG}}(f) \in \mathcal{G}(\sigma_{\text{ALG}}) \iff \phi_{\text{ALG}}(f) \in \mathcal{G}(\nu_{\text{ALG}}) \quad \text{for} \quad \sigma_{\text{ALG}} = n \cdot \nu_{\text{ALG}}.$$

The following standard lemma establishes that the sub-Gaussian property is closed under scaling and summation.

Lemma 12 (Summation and scaling preserve sub-Gaussianity) *Suppose $G_1 \in \mathcal{G}(\sigma_1)$. Then, for all $\beta \in \mathbb{R}$, we have $\beta \cdot G_1 \in \mathcal{G}(\beta\sigma_1)$. Furthermore, if G_1 is $\mathcal{F}$-measurable and $G_2 \in \mathcal{G}(\sigma_2)$ given $\mathcal{F}$, then $G_1 + G_2 \in \mathcal{G}(\sqrt{\sigma_1^2 + \sigma_2^2})$.*

Proof Fix any $\beta \in \mathbb{R}$. Since $G_1 \in \mathcal{G}(\sigma_1)$, for each $\lambda \in \mathbb{R}$,

$$\mathbb{E}\big[\exp(\lambda \cdot \beta \cdot G_1)\big] \leq \exp\Big(\frac{\lambda^2(\beta\sigma_1)^2}{2}\Big),$$

so that $\beta G_1 \in \mathcal{G}(\beta\sigma_1)$ as advertised.

Furthermore, if G_1 is $\mathcal{F}$-measurable and $G_2 \in \mathcal{G}(\sigma_2)$ given $\mathcal{F}$, then, for each $\lambda \in \mathbb{R}$,

$$\begin{aligned}
\mathbb{E}\Big[\exp\big(\lambda \cdot (G_1 + G_2)\big)\Big] &= \mathbb{E}\big[\exp(\lambda \cdot G_1 + \lambda \cdot G_2)\big] \\
&= \mathbb{E}\Big[\exp(\lambda \cdot G_1) \cdot \mathbb{E}\big[\exp(\lambda \cdot G_2) \mid \mathcal{F}\big]\Big] \\
&\leq \exp\Big(\frac{\lambda^2\sigma_2^2}{2}\Big) \cdot \mathbb{E}\Big[\exp\big(\lambda \cdot f(G_2)\big)\Big] \\
&= \exp\Big(\frac{\lambda^2\sigma_1^2}{2}\Big) \cdot \exp\Big(\frac{\lambda^2\sigma_2^2}{2}\Big) = \exp\Big(\frac{\lambda^2\big(\sigma_1^2+\sigma_2^2\big)}{2}\Big),
\end{aligned}$$

so that $G_1 + G_2 \in \mathcal{G}(\sqrt{\sigma_1^2 + \sigma_2^2})$ as claimed. ■

Appendix B. KT-SPLIT Discussion

Many thinning algorithms offer high-probability bounds on the integration error $|\mathbb{P}_{\mathcal{S}_{\text{in}}} f - \mathbb{P}_{\mathcal{S}_{\text{ALG}}} f|$. We capture such bounds abstractly using the following definition of a sub-Gaussian thinning algorithm.

Definition 13 (Sub-Gaussian thinning algorithm) *For a function f, we call a thinning algorithm* ALG f-sub-Gaussian *with parameter ν and write* $\text{ALG} \in \mathcal{G}^f(\nu)$ *if*

$$\mathbb{E}[\exp(\lambda(\mathbb{P}_{\mathcal{S}_{\text{in}}} f - \mathbb{P}_{\mathcal{S}_{\text{ALG}}} f)) \mid \mathcal{S}_{\text{in}}] \leq \exp\Big(\frac{\lambda^2\nu^2(n)}{2}\Big) \quad \textit{for all} \quad \lambda \in \mathbb{R}.$$

Thm. 13 is equivalent to a sub-Gaussian tail bound for the integration error, and, by Boucheron et al. (2013, Section 2.3), if ALG $\in \mathcal{G}^f(\nu)$ then $\mathbb{E}[\mathbb{P}_{\mathcal{S}_{\text{ALG}}} f \mid \mathcal{S}_{\text{in}}] = \mathbb{P}_{\mathcal{S}_{\text{in}}} f$ and, for all $\delta \in (0, 1)$,

$$|\mathbb{P}_{\mathcal{S}_{\text{in}}} f - \mathbb{P}_{\mathcal{S}_{\text{ALG}}} f| \leq \nu(n)\sqrt{2\log(\tfrac{2}{\delta})}, \text{ with probability at least } 1-\delta \text{ given } \mathcal{S}_{\text{in}}.$$

Hence the integration error of ALG is dominated by the sub-Gaussian parameter $\nu(n)$.

Example 1 (KT-SPLIT) Given a kernel $\mathbf{k}$ and n input points $\mathcal{S}_{\text{in}}$, the KT-SPLIT(δ) algorithm[1] of Dwivedi and Mackey (2021a,b, Alg. 1a) takes $\Theta(n^2)$ kernel evaluations to output a coreset of size n_{out} with better-than-i.i.d. integration error. Specifically, Dwivedi and Mackey (2021a, Thm. 1) prove that, on an event with probability $1-\delta$, KT-SPLIT$(\delta) \in \mathcal{G}^f(\nu)$ with

$$\nu(n) = \tfrac{2}{n_{\text{out}}\sqrt{3}}\sqrt{\log(\tfrac{4(n-n_{\text{out}})}{\delta})\|\mathbf{k}\|_\infty}$$

for all f with $\|f\|_{\mathbf{k}} = 1$. ■

Example 2 (Kernel thinning) Given a kernel $\mathbf{k}$ and n input points $\mathcal{S}_{\text{in}}$, the generalized kernel thinning (KT(δ)) algorithm[1] of Dwivedi and Mackey (2021a,b, Alg. 1) takes $\Theta(n^2)$ kernel evaluations to output a coreset of size n_{out} with near-optimal MMD error. In particular, by leveraging an appropriate auxiliary kernel $\mathbf{k}_{\text{split}}$, Dwivedi and Mackey (2021a, Thms. 2-4) establish that, on an event with probability $1-\delta$, KT$(\delta) \in \mathcal{G}_{\mathbf{k}}(a, v)$ with

$$a_n = \tfrac{C_a}{n_{\text{out}}}\sqrt{\|\mathbf{k}_{\text{split}}\|_\infty}, \quad \text{and} \quad v_n = \tfrac{C_v}{n_{\text{out}}}\sqrt{\|\mathbf{k}_{\text{split}}\|_\infty \log(\tfrac{4(n-n_{\text{out}})}{\delta})}\,\mathfrak{M}_{\mathcal{S}_{\text{in}},\mathbf{k}_{\text{split}}}, \tag{9}$$

where $\|\mathbf{k}_{\text{split}}\|_\infty = \sup_x \mathbf{k}_{\text{split}}(x, x)$, C_a and C_v are explicit constants, and $\mathfrak{M}_{\mathcal{S}_{\text{in}},\mathbf{k}_{\text{split}}} \geq 1$ is non-decreasing in n and varies based on the tails of $\mathbf{k}_{\text{split}}$ and the radius of the ball containing $\mathcal{S}_{\text{in}}$. ■

Appendix C. Proof of Thm. 3: Runtime of COMPRESS

First, we bound the running time of COMPRESS. By definition, COMPRESS makes four recursive calls to COMPRESS on inputs of size $n/4$. Then, HALVE is run on an input of size $2^{g+1}\sqrt{n}$. Thus, r_{C} satisfies the recursion

$$r_{\text{C}}(n) = 4r_{\text{C}}\left(\tfrac{n}{4}\right) + r_{\text{H}}(\sqrt{n}2^{g+1}).$$

Since $r_{\text{C}}(4^g) = 0$, we may unroll the recursion to find that

$$r_{\text{C}}(n) = \textstyle\sum_{i=0}^{\beta_n} 4^i r_{\text{H}}(2^{g+1}\sqrt{n4^{-i}}),$$

as claimed in (3).

Appendix D. Proof of Thm. 5: MMD guarantees for COMPRESS

Our proof proceeds in several steps. To control the MMD (1), we need to control the Hilbert norm of the measure discrepancy of COMPRESS (8), which we first write as a weighted sum of measure discrepancies from different (conditionally independent) runs of HALVE. To effectively leverage the MMD tail bound assumption for this weighted sum, we reduce the problem to establishing a concentration inequality for the operator norm of an associated matrix. We carry out this plan in four steps summarized below.

1. The δ argument indicates that each input parameter $\delta_i = \frac{\delta}{\ell - n_{\text{out}}}$ in Dwivedi and Mackey (2021a, Alg. 1a), where ℓ is the size of the input to KT-SPLIT(δ) or KT(δ) and n_{out} is the target output size.

First, we express the MMD associated with each HALVE measure discrepancy as the Euclidean norm of a suitable vector (Thm. 14). Second, we define a matrix dilation operator for a vector that allows us to control vector norms using matrix spectral norms (Thm. 15). Third, we establish moment bounds for these matrices by leveraging tail bounds for the MMD error (Thm. 17). Finally, we prove and apply a subexponential matrix Freedman concentration inequality (Thm. 16) to control the MMD error for the COMPRESS output.

We now begin our formal argument. We will make use of the unrolled representation for the COMPRESS measure discrepancy $\psi_{\mathrm{C}}(\mathcal{S}_{\mathrm{in}})$ in terms of the HALVE inputs $(\mathcal{S}^{\mathrm{in}}_{k,j})_{j\in[4^k]}$ of size $n_k = 2^{\mathfrak{g}+1-k}\sqrt{n}$ for $0 \leq k \leq \log_4 n - \mathfrak{g} - 1$. For brevity, we will use the shorthand $\psi_{\mathrm{C}} \triangleq \psi_{\mathrm{C}}(\mathcal{S}_{\mathrm{in}})$, $\psi^{\mathrm{H}}_{k,j} \triangleq \psi_{\mathrm{H}}(\mathcal{S}^{\mathrm{in}}_{k,j})$, and $\psi_{\mathrm{T}} \triangleq \psi_{\mathrm{T}}(\mathcal{S}_{\mathrm{C}})$ hereafter.

D.1. Reducing MMD to vector Euclidean norm

Number the elements of $\mathcal{S}_{\mathrm{in}}$ as $(x_1, \ldots, x_n)$, define the $n \times n$ kernel matrix $\mathbf{K} \triangleq (\mathbf{k}(x_i, x_j))_{i,j=1}^n$, and let $\mathbf{K}^{\frac{1}{2}}$ denote a matrix square-root such that $\mathbf{K} = \mathbf{K}^{\frac{1}{2}} \cdot \mathbf{K}^{\frac{1}{2}}$ (which exists since $\mathbf{K}$ is a positive semidefinite matrix for any kernel $\mathbf{k}$). Next, let $\mathcal{S}^{\mathrm{out}}_{k,j}$ denote the output sequence corresponding to $\psi^{\mathrm{H}}_{k,j}$ (i.e., running HALVE on $\mathcal{S}^{\mathrm{in}}_{k,j}$), and let $\{e_i\}_{i=1}^n$ denote the canonical basis of $\mathbb{R}^n$. The next lemma (with proof in App. D.5) relates the Hilbert norms to Euclidean norms of carefully constructed vectors.

Lemma 14 (MMD as a vector norm) *Define the vectors*

$$u_{k,j} \triangleq \mathbf{K}^{\frac{1}{2}} \sum_{i=1}^n e_i \Big(\mathbf{1}(x_i \in \mathcal{S}^{\mathrm{in}}_{k,j}) - 2 \cdot \mathbf{1}(x_i \in \mathcal{S}^{\mathrm{out}}_{k,j}) \Big), \text{ and } u_{\mathrm{C}} \triangleq \sum_{k=0}^{\log_4 n - \mathfrak{g} - 1} \sum_{j=1}^{4^k} w_{j,k} u_{j,k}, \tag{10}$$

where $w_{j,k} \triangleq \frac{\sqrt{n}}{2^{\mathfrak{g}+1+k}}$. *Then, we have*

$$n^2 \cdot \mathrm{MMD}^2_{\mathbf{k}}(\mathcal{S}_{\mathrm{in}}, \mathcal{S}_{\mathrm{C}}) = \|u_{\mathrm{C}}\|_2^2, \quad \text{and} \tag{11}$$

$$\mathbb{E}[u_{k,j} | \{u_{k',j'}, j' \in [4^{k'}], k' > k\}] = 0 \quad \text{for} \quad k = 0, \ldots, \log_4 n - \mathfrak{g} - 2, \tag{12}$$

and $u_{k,j}$ *for* $j \in [4^k]$ *are conditionally independent given* $\{u_{k',j'}, j' \in [4^{k'}], k' > k\}$.

Applying (11), we effectively reduce the task of controlling the MMD errors to controlling the Euclidean norm of suitably defined vectors. Next, we reduce the problem to controlling the spectral norm of a suitable matrix.

D.2. Reducing vector Euclidean norm to matrix spectral norm

To this end, we define the following symmetric dilation matrix operator: given a vector $u \in \mathbb{R}^n$, define the matrix $\mathbf{M}_u$ as

$$\mathbf{M}_u \triangleq \begin{pmatrix} 0 & u^\top \\ u & \mathbf{0}_{n \times n} \end{pmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}. \tag{13}$$

It is straightforward to see that $u \mapsto \mathbf{M}_u$ is a linear map. In addition, the matrix $\mathbf{M}_u$ also satisfies a few important properties (established in App. D.6) that we use in our proofs.

Lemma 15 (Properties of the dilation operator) *For any* $u \in \mathbb{R}^n$, *the matrix* $\mathbf{M}_u$ (13) *satisfies*

$$\|\mathbf{M}_u\|_{\mathrm{op}} \overset{(a)}{=} \|u\|_2 \overset{(b)}{=} \lambda_{\max}(\mathbf{M}_u), \quad \text{and} \quad \mathbf{M}_u^q \overset{(c)}{\preceq} \|u\|_2^q \mathbf{I}_{n+1} \text{ for all } q \in \mathbb{N}. \tag{14}$$

Define the shorthand $\mathbf{M}_{k,j} \triangleq \mathbf{M}_{w_{k,j} u_{k,j}}$ (defined in Thm. 14). Applying Thms. 14 and 15, we find that

$$n \, \mathrm{MMD}_{\mathbf{k}}(\mathcal{S}_{\mathrm{in}}, \mathcal{S}_{\mathrm{C}}) \overset{(11)}{=} \|u_{\mathrm{C}}\|_2 \overset{(14)}{=} \lambda_{\max}(\mathbf{M}_{u_{\mathrm{C}}}) \overset{(i)}{=} \lambda_{\max}\Big(\sum_{k=0}^{\log_4 n - \mathfrak{g} - 1} \sum_{j=1}^{4^k} \mathbf{M}_{k,j} \Big), \tag{15}$$

where step (i) follows from the linearity of the dilation operator. Thus to control the MMD error, it suffices to control the maximum eigenvalue of the sum of matrices appearing in (15).

D.3. Controlling matrix spectral norm via a matrix Freedman inequality

To control the maximum eigenvalue of the matrix $\mathbf{M}_{uc}$, we make use of (15) and the following subexponential generalization of the matrix Freedman inequality of Tropp (2011, Thm. 1.2). The proof of Thm. 16 can be found in App. D.7. For two matrices A and B of the same size, we write $A \preceq B$ if $B - A$ is positive semidefinite.

Lemma 16 (Subexponential matrix Freedman inequality) *If a finite sequence $\{\mathbf{Y}_i\}_{i=1}^N$ of symmetric random matrices in $\mathbb{R}^{m\times m}$ satisfies*

$$\mathbb{E}\Big[\mathbf{Y}_i \big| \{\mathbf{Y}_j\}_{j=1}^{i-1}\Big] = 0 \quad \textit{and} \quad \mathbb{E}\Big[\mathbf{Y}_i^p \big| \{\mathbf{Y}_j\}_{j=1}^{i-1}\Big] \preceq \tfrac{p!}{2} R^{p-2} \mathbf{A}_i^2, \textit{ for all } i \in [N], \textit{ and } p \in \mathbb{N}\backslash\{1\}, \tag{16}$$

for some $R > 0$, and a deterministic sequence $\{\mathbf{A}_i\}_{i=1}^N$ of symmetric matrices in $\mathbb{R}^{m\times m}$, then

$$\mathbb{P}[\lambda_{\max}(\textstyle\sum_{i=1}^N \mathbf{Y}_i) \geq \sqrt{2\sigma^2(\log(m)+t)} + R(\log(m)+t)] \leq e^{-t} \quad \textit{for all} \quad t > 0$$

and equivalently

$$\mathbb{P}[\lambda_{\max}(\textstyle\sum_{i=1}^N \mathbf{Y}_i) \leq \sqrt{2\sigma^2 \log(m/\delta)} + R\log(m/\delta)] \geq 1-\delta \quad \textit{for all} \quad \delta \in (0,1).$$

To apply Thm. 16 with the matrices $\{\mathbf{M}_{k,j}\}$, we need to establish the zero-mean and moment bound conditions for suitable R and $\mathbf{A}_{k,j}$ in (16).

D.3.1. VERIFYING THE ZERO MEAN CONDITION

To this end, first we note that the conditional independence and zero-mean property of $\left\{\psi_{k,j}^{\mathrm{H}}\right\}$ implies that the random vectors $u_{k,j}$ and the matrices $\mathbf{M}_{k,j}$ also satisfy a similar property, and in particular that

$$\mathbb{E}\left[\mathbf{M}_{k,j} \middle| \left\{\mathbf{M}_{k',j'}, k' > k, j' \in [4^{k'}]\right\}\right] = \mathbf{0} \quad \text{for} \quad j \in [4^k], k \in \{0,1,\ldots,\log_4 n - \mathfrak{g} - 1\}. \tag{17}$$

D.3.2. ESTABLISHING MATRIX MOMENT BOUNDS VIA MMD TAIL BOUNDS FOR HALVE

To establish the moment bounds on $\mathbf{M}_{k,j}$, note that Thms. 14 and 15 imply that

$$\mathbf{M}_{k,j}^q = \mathbf{M}_{w_{k,j}u_{k,j}}^q \overset{(14)}{\preceq} \left\|w_{k,j}u_{k,j}\right\|_2^q \cdot \mathbf{I}_{n+1} \overset{(11)}{=} w_{k,j}^q \left\|u_{k,j}\right\|_2^q \cdot \mathbf{I}_{n+1} \tag{18}$$

where $w_{k,j}$ was defined in Thm. 14. Thus it suffices to establish the moment bounds on $\|u_{k,j}\|_{\mathbf{k}}^q$. To this end, we first state a lemma that converts tail bounds to moment bounds (see App. D.8 for a proof inspired by Boucheron et al. (2013, Thm. 2.3)).

Lemma 17 (Tail bounds imply moment bounds) *For a non-negative random variable Z,*

$$\mathbb{P}[Z > a + v\sqrt{t} + ct] \leq e^{-t},\ \forall t \geq 0 \implies \mathbb{E}[Z^q] \leq \tfrac{q!}{2}(2a+2c+2v)^q,\ \forall q \in \mathbb{N} \textit{ with } q \geq 2.$$

To obtain a moment bound for $\left\|u_{k,j}\right\|_2$, we first state some notation. For each n, define the quantities

$$a_n' \triangleq n a_n, \quad v_n' \triangleq n v_n \tag{19}$$

where a_n and v_n are the parameters such that $\textsc{Halve} \in \mathcal{G}_{\mathbf{k}}(a_n, v_n)$ on inputs of size n. Using an argument similar to Thm. 14, we have

$$\left\|u_{k,j}\right\|_2 = n_{k,j}\, \mathrm{MMD}_{\mathbf{k}}(\mathcal{S}_{k,j}^{\mathrm{in}}, \mathcal{S}_{k,j}^{\mathrm{out}}) \quad \text{for} \quad n_{k,j} = |\mathcal{S}_{k,j}^{\mathrm{in}}| = \sqrt{n} 2^{\mathfrak{g}+1-k}.$$

Thereby, using the $\mathcal{G}_{\mathbf{k}}$ assumption on HALVE implies that

$$\mathbb{P}[\left\|u_{k,j}\right\|_2 \geq a'_{\ell_k} + v'_{\ell_k}\sqrt{t}] \leq e^{-t} \text{ for all } t \geq 0, \text{ where } \ell_k \triangleq n_{k,j} = \sqrt{n}2^{\mathfrak{g}+1-k}, \tag{20}$$

conditioned on $\{u_{k',j'}, j' \in [4^{k'}], k' > k\}$. Combining the bound (20) with Thm. 17 yields that

$$\mathbb{E}[\left\|u_{k,j}\right\|_2^q | \{u_{k',j'}, j' \in [4^{k'}], k' > k\}] \leq \tfrac{q!}{2}(2a'_{\ell_k} + 2v'_{\ell_k})^q, \tag{21}$$

for all $q \in \mathbb{N}$, where ℓ_k is defined in (20). Now, putting together (18) and (21), and using the conditional independence of $\mathbf{M}_{k,j}$, we obtain the following control on the q-th moments of $\mathbf{M}_{k,j}$ for $q \geq 2$:

$$\begin{aligned}
\mathbb{E}\left[\mathbf{M}_{k,j}^q \middle| \left\{\mathbf{M}_{k',j'}, k' > k, j' \in [4^{k'}]\right\}\right] &\overset{(18)}{\preceq} w_{k,j}^q \cdot \mathbb{E}\left[\left\|u_{k,j}\right\|_2^q \middle| \left\{u_{k',j'}, k' > k, j' \in [4^{k'}]\right\}\right] \cdot \mathbf{I}_{n+1} \\
&\overset{(21)}{\preceq} w_{k,j}^q \cdot \left[\tfrac{q!}{2}(2a'_{\ell_k} + 2v'_{\ell_k})^q\right] \cdot \mathbf{I}_{n+1} \\
&= \tfrac{q!}{2} R_{k,j}^{q-2} \mathbf{A}_{k,j}^2 \text{ where } \begin{matrix} R_{k,j} \triangleq 2w_{k,j}(a'_{\ell_k} + v'_{\ell_k}) \\ \mathbf{A}_{k,j} \triangleq R_{k,j}\mathbf{I}_{n+1}, \end{matrix}
\end{aligned} \tag{22}$$

where ℓ_k is defined in (20).

D.4. Putting the pieces together

Putting (17) and (22) together, we conclude that with a suitable ordering of the indices (k, j), the assumptions of Thm. 16 are satisfied by the random matrices $\left\{\mathbf{M}_{k,j}, j \in [4^k], k \in \{0, 1, \ldots, \log_4 n - \mathfrak{g} - 1\}\right\}$, the deterministic sequence $\left\{\mathbf{A}_{k,j}\right\}$, and any scalar R satisfying $R \geq \max_{k,j} R_{k,j}$. Now, since $\ell_k = \sqrt{n}2^{\mathfrak{g}+1-k}$ is decreasing in k, $w_{k,j} = \frac{\ell_k}{4^{\mathfrak{g}+1}}$, and na'_n and nv'_n, (19) are assumed non-decreasing in n, we find that $R = n \cdot R_{\mathrm{M,C}}$ is a valid choice as

$$\begin{aligned}
n \cdot R_{\mathrm{M,C}} &= n\, 2(a_{\sqrt{n}2^{\mathfrak{g}+1}} + v_{\sqrt{n}2^{\mathfrak{g}+1}}) \\
&= \tfrac{2\sqrt{n}}{2^{\mathfrak{g}+1}}(a'_{\sqrt{n}2^{\mathfrak{g}+1}} + v'_{\sqrt{n}2^{\mathfrak{g}+1}}) \\
&= 2\tfrac{\ell_0}{4^{\mathfrak{g}+1}}(a'_{\ell_0} + v'_{\ell_0}) \\
&\geq \max_{k \leq \log_4 n - \mathfrak{g} - 1} 2\tfrac{\ell_k}{4^{\mathfrak{g}+1}}(a'_{\ell_k} + v'_{\ell_k}) \\
&= \max_{k,j} 2w_{k,j}(a'_{\ell_k} + v'_{\ell_k}) = \max_{k,j} R_{k,j}.
\end{aligned}$$

Moreover, since a'_n and v'_n, are assumed non-decreasing in n, ℓ_k is decreasing in k, and $w_{k,j} = \frac{\sqrt{n}}{2^{\mathfrak{g}+1+k}}$, our choice $\sigma^2_{\mathrm{M,C}}$ satisfies

$$\begin{aligned}
n^2 \cdot \sigma^2_{\mathrm{M,C}} &= n^2(\log_4 n - \mathfrak{g})R^2_{\mathrm{M,C}} \\
&= n^2(\log_4 n - \mathfrak{g})(2(a_{\sqrt{n}2^{\mathfrak{g}+1}} + c_{\sqrt{n}2^{\mathfrak{g}+1}} + \sqrt{2}v_{\sqrt{n}2^{\mathfrak{g}+1}}))^2 \\
&= (\log_4 n - \mathfrak{g})\tfrac{n}{4^{\mathfrak{g}+1}}(2(a'_{\ell_0} + v'_{\ell_0}))^2 \\
&\geq \textstyle\sum_{k=0}^{\log_4 n - \mathfrak{g} - 1} \tfrac{n}{4^{\mathfrak{g}+1}}(2(a'_{\ell_k} + v'_{\ell_k}))^2 \\
&= \textstyle\sum_{k=0}^{\log_4 n - \mathfrak{g} - 1} \sum_{j=1}^{4^k} \tfrac{n}{4^{\mathfrak{g}+1+k}}(2(a'_{\ell_k} + v'_{\ell_k}))^2 \\
&= \textstyle\sum_{k=0}^{\log_4 n - \mathfrak{g} - 1} \sum_{j=1}^{4^k} (2w_{k,j}(a'_{\ell_k} + v'_{\ell_k}))^2 \\
&= \textstyle\sum_{k=0}^{\log_4 n - \mathfrak{g} - 1} \sum_{j=1}^{4^k} R^2_{k,j} \overset{(22)}{=} \left\|\sum_{k=0}^{\log_4 n - \mathfrak{g} - 1} \sum_{j=1}^{4^k} \mathbf{A}^2_{k,j}\right\|_{\mathrm{op}}.
\end{aligned}$$

Finally, applying (15) along with Thm. 16, we conclude that

$$\begin{aligned}
&\mathbb{P}[\mathrm{MMD}(\mathcal{S}_{\mathrm{in}}, \mathcal{S}_{\mathrm{C}}) \geq \sigma_{\mathrm{M,C}}\sqrt{2(\log(n+1)+t)} + R_{\mathrm{M,C}}(\log(n+1)+t)] \\
&= \mathbb{P}[\lambda_{\max}(\textstyle\sum_{k=0}^{\log_4 n - \mathfrak{g} - 1} \sum_{j=1}^{4^k} \mathbf{M}_{k,j}) \geq n\sigma_{\mathrm{M,C}}\sqrt{2(\log(n+1)+t)} + nR_{\mathrm{M,C}}(\log(n+1)+t)] \\
&\leq e^{-t} \quad \text{for all} \quad t > 0.
\end{aligned}$$

D.5. Proof of Thm. 14: MMD as a vector norm

Let $v_{k,j} \triangleq \sum_{i=1}^n e_i \Big(\mathbf{1}(x_i \in \mathcal{S}_{k,j}^{\mathrm{in}}) - 2 \cdot \mathbf{1}(x_i \in \mathcal{S}_{k,j}^{\mathrm{out}}) \Big)$. By the reproducing property of $\mathbf{k}$ we have

$$\begin{aligned}
\|\psi_{k,j}^{\mathrm{H}}(\mathbf{k})\|_{\mathbf{k}}^2 &= \Big\| \textstyle\sum_{x \in \mathcal{S}_{k,j}^{\mathrm{in}}} \mathbf{k}(x, \cdot) - 2 \sum_{x \in \mathcal{S}_{k,j}^{\mathrm{out}}} \mathbf{k}(x, \cdot) \Big\|_{\mathbf{k}}^2 \\
&= \textstyle\sum_{x \in \mathcal{S}_{k,j}^{\mathrm{in}}, y \in \mathcal{S}_{k,j}^{\mathrm{in}}} \mathbf{k}(x,y) - 2 \sum_{x \in \mathcal{S}_{k,j}^{\mathrm{out}}, y \in \mathcal{S}_{k,j}^{\mathrm{in}}} \mathbf{k}(x,y) + \sum_{x \in \mathcal{S}_{k,j}^{\mathrm{out}}, y \in \mathcal{S}_{k,j}^{\mathrm{out}}} \mathbf{k}(x,y) \\
&= v_{k,j}^\top \mathbf{K} v_{k,j} \overset{(10)}{=} \big\| u_{k,j} \big\|_2^2. \qquad (23)
\end{aligned}$$

Using (10) and (13), and mimicking the derivation above (23), we can also conclude that

$$\|\psi_{\mathrm{C}}(\mathbf{k})\|_{\mathbf{k}}^2 = \|u_{\mathrm{C}}\|_2^2.$$

Additionally, we note that

$$\mathrm{MMD}_{\mathbf{k}}(\mathcal{S}_{\mathrm{in}}, \mathcal{S}_{\mathrm{C}}) = \sup_{\|f\|_{\mathbf{k}} = 1} \tfrac{1}{n} \big\langle f, \psi_{\mathrm{C}}(\mathbf{k}) \big\rangle_{\mathcal{H}_{\mathbf{k}}} = \tfrac{1}{n} \|\psi_{\mathrm{C}}(\mathbf{k})\|_{\mathbf{k}}.$$

Finally the conditional independence and zero mean property (12) follows by noting that conditioned on $(\mathcal{S}_{i',j'}^{\mathrm{in}})_{i' > i, j' \geq 1}$, the sets $(\mathcal{S}_{i,j}^{\mathrm{in}})_{j \geq 1}$ are independent.

D.6. Proof of Thm. 15: Properties of the dilation operator

For claim (a) in the display (14), we have

$$\mathbf{M}_u^2 = \begin{pmatrix} \|u\|_2^2 & \mathbf{0}_n^\top \\ \mathbf{0}_n & uu^\top \end{pmatrix} \overset{(i)}{\preceq} \|u\|_2^2 \mathbf{I}_{n+1} \quad \Longrightarrow \quad \|\mathbf{M}_u\|_{\mathrm{op}} \overset{(ii)}{=} \|u\|_2,$$

where step (i) follows from the standard fact that $uu^\top \preceq \|u\|_2^2 \mathbf{I}_n$ and step (ii) from the facts $\mathbf{M}_u^2 \widetilde{e}_1 = \|u\|_2^2 \widetilde{e}_1$ for $\widetilde{e}_1$ the first canonical basis vector of $\mathbb{R}^{n+1}$ and $\|\mathbf{M}_u\|_{\mathrm{op}}^2 = \big\|\mathbf{M}_u^2\big\|_{\mathrm{op}}$. Claim (b) follows directly by verifying that the vector $v = [1, \frac{u^\top}{\|u\|_2}]^\top$ is an eigenvector of $\mathbf{M}_u$ with eigenvalue $\|u\|_2$. Finally, claim (c) follows directly from the claim (a) and the fact that $\|\mathbf{M}_u^q\|_{\mathrm{op}} = \|\mathbf{M}_u\|_{\mathrm{op}}^q$ for all integers $q \geq 1$.

D.7. Proof of Thm. 16: Subexponential matrix Freedman inequality

We first note the following lemmas about the moment generating functions (MGFs) for matrix valued random variables.

Lemma 18 (Matrix tail bounds (Tropp, 2011, Thm 2.3)) *Let $\mathbf{X}_k$ be an adaptive sequence of matrix valued random variables in $\mathbb{R}^{m \times m}$ (with respect to the filtration $\mathcal{F}_k$ and let $\mathbf{V}_k$ be a sequence that is measureable with respect to $\mathcal{F}_{k-1}$. Suppose that, for some $\theta > 0$,*

$$\log \mathbb{E}\big[\exp(\theta \mathbf{X}_k) | \mathcal{F}_{k-1}\big] \preceq g(\theta) \mathbf{V}_k.$$

Let $\mathbf{Y}_k = \sum_{i \leq k} \mathbf{X}_i$ and $\mathbf{W}_k = \sum_{i \leq k} \mathbf{V}_i$. Then,

$$\mathbb{P}\big[\exists k > 0 : \lambda_{\max}(\mathbf{Y}_k) \geq t \text{ and } \lambda_{\max}(\mathbf{W}_k) \leq w\big] \leq m \exp\big(-\theta t + g(\theta) w\big).$$

Lemma 19 (Supexponential matrix MGF) *Let* $\mathbf{X}$ *be a random symmetric matrix such that*

$$\mathbb{E}\mathbf{X} = 0 \quad \textit{and} \quad \mathbb{E}\mathbf{X}^q \preceq \tfrac{q!}{2} R^{q-2}\mathbf{A}^2 \quad \textit{for} \quad q = 2, 3, \ldots.$$

Then,

$$\mathbb{E}\exp(\theta\mathbf{X}) \preceq \exp\Big(\tfrac{\theta^2}{2(1-R\theta)} \cdot \mathbf{A}^2\Big) \quad \textit{for all} \quad \theta \in (0, 1/R).$$

Proof When $R = 1$, the result coincides with Tropp (2012, Lem. 6.8). To obtain the general result, we apply Tropp (2012, Lem. 6.8) to $\mathbf{X}' = \mathbf{X}/R$ and $\mathbf{A}' = \mathbf{A}/R$. ■

We apply Thm. 19 conditional on $\{\mathbf{X}_i\}_{i<k}$ along with the operator monotonicity of $\log$ to get

$$\log \mathbb{E}\Big[\exp(\theta\mathbf{X}_k)|\{\mathbf{X}_i\}_{i<k}\Big] \preceq \tfrac{\theta^2}{2(1-R\theta)} \cdot A^2 \quad \text{for all} \quad \theta \in (0, 1/R).$$

By assumption, we have that $\lambda_{\max}(\sum_i \mathbf{A}_i^2) \le \sigma^2$. Thus, applying Thm. 18, we get, for all $t > 0$,

$$\mathbb{P}[\lambda_{\max}(\textstyle\sum_i \mathbf{X}_i) \ge t] \le m \inf_{\theta \in (0,1/R)} \exp\Big(-\theta t + \tfrac{\theta^2\sigma^2}{2(1-R\theta)}\Big) = m\exp(-\tfrac{\sigma^2}{R^2} h_1(Rt/\sigma^2))$$

where, by Boucheron et al. (2013, Sec. 2.4),

$$h_1(u) \triangleq 1 + u - \sqrt{1+2u} \quad \text{for} \quad u > 0.$$

Now, since the inverse of h_1 takes the form $h_1^{-1}(u) = u + \sqrt{2u}$ for $u > 0$ by Boucheron et al. (2013, Sec. 2.4), we additionally have

$$\begin{aligned}&\mathbb{P}[\lambda_{\max}(\textstyle\sum_i \mathbf{X}_i) \ge \tfrac{\sigma^2}{R} h_1^{-1}(\tfrac{R^2}{\sigma^2}(\log(m) + t))] \\ &= \mathbb{P}[\lambda_{\max}(\textstyle\sum_i \mathbf{X}_i) \ge \sqrt{2\sigma^2(\log(m)+t)} + R(\log(m)+t)] \le \exp(-t)\end{aligned}$$

as advertised.

D.8. Proof of Thm. 17: Tail bounds imply moment bounds

Let $\bar{a} = a + v/2$ and $\bar{c} = c + v/2$. For all $t > 0$, by the arithmetic-geometric mean inequality,

$$\mathbb{P}[Z > \bar{a} + \bar{c}t] = \mathbb{P}[Z > a + v/2 + (c + v/2)t] \le \mathbb{P}[Z > a + v\sqrt{t} + ct] \le e^{-t}.$$

We begin by bounding the moments of the shifted random variable $X = Z - \bar{a}$. First note that $X = X_+ - X_-$ where $X_\pm = \max(\pm X, 0)$ and that $|X|^q = X_+^q + X_-^q$. Furthermore, $X_-^q \le \bar{a}^q$ by the nonnegativity of Z, so that $|X|^q \le \bar{a}^q + X_+^q$. Since $\mathbb{P}[X_+ > u] = \mathbb{P}[X > u] = \mathbb{P}[Z > \bar{a} + u]$ for any $u > 0$ we will use the our tail bounds to control the moments of X_+. In particular, we have

$$\begin{aligned}\mathbb{E}\big[X_+^q\big] &\overset{(i)}{=} q\textstyle\int_0^\infty u^{q-1}\mathbb{P}[X_+ > u]du \\ &\overset{(ii)}{=} q\bar{c}\textstyle\int_0^\infty (\bar{c}t)^{q-1}\mathbb{P}[X_+ > \bar{c}t]dt \\ &\overset{(iii)}{\le} q\bar{c}^q\textstyle\int_0^\infty t^{q-1}e^{-t}dt \overset{(iv)}{=} \Gamma(q)q\,\bar{c}^q = q!\,\bar{c}^q.\end{aligned}$$

where we have applied (i) integration by parts, (ii) the substitution $u = \bar{c}t$, (iii) the assumed tail bound for Z, and (iv) the definition of the Gamma function.

Now since $Z = X + \bar{a}$, we have $\mathbb{E}Z^q \le 2^{q-1}(\bar{a}^q + \mathbb{E}|X|^q) \le 2^{q-1}(2\bar{a}^q + \mathbb{E}X_+^q)$ from the convexity of t^q. For each $q \ge 2$, our combined results now yield

$$\begin{aligned}\mathbb{E}Z^q &\le (2\bar{a})^q + \tfrac{q!}{2}(2\bar{c})^q \\ &\le \tfrac{q!}{2}(2\bar{a})^q + \tfrac{q!}{2}(2\bar{c})^q \\ &\le \tfrac{q!}{2}(2\bar{a} + 2\bar{c})^q = \tfrac{q!}{2}(2a + 2c + 2v)^q\end{aligned}$$

since $x^q + y^q \le (x+y)^q$ for all $q \in \mathbb{N}$ and $x, y \ge 0$.

Appendix E. COMPRESS MMD Examples

Remark 20 (Symmetrization) We can convert any halving algorithm into one that satisfies the unbiasedness condition $\mathbb{E}\left[\mathbb{P}_{\text{HALVE}}\mathbf{k} \mid \mathcal{S}_{\text{in}}\right] = \mathbb{P}_{\text{in}}\mathbf{k}$ without impacting integration error by *symmetrization*, i.e., by returning either the outputted half or its complement with equal probability.

Example 3 (KT-COMPRESS) Consider running COMPRESS with, for each HALVE input of size ℓ, HALVE $=$ KT$(\frac{\ell}{2n-\ell_n}\delta)$ from Ex. 2 after symmetrizing as in Thm. 20. Since KT has $\Theta(n^2)$ runtime, COMPRESS yields near-linear $\mathcal{O}(n \log n)$ runtime by Thm. 4. Moreover, as we detail in App. I.1, on an event of probability at least $1-\delta$, every HALVE call invoked by COMPRESS is $\mathbf{k}$-sub-Gaussian with

$$a_\ell = \tfrac{2C_a}{\ell}\sqrt{\|\mathbf{k}_{\text{split}}\|_\infty}, \quad \text{and} \quad v_\ell = \tfrac{2C_v}{\ell}\sqrt{\|\mathbf{k}_{\text{split}}\|_\infty \log(\tfrac{4n-2\ell_n}{\delta})}\,\mathfrak{M}_{\mathcal{S}_{\text{in}},\mathbf{k}_{\text{split}}},$$

in the notation of Ex. 2. Hence, Thm. 6 implies that, on the same event, KT-COMPRESS has $\mathbf{k}$-sub-Gaussian error $\varepsilon_{\mathbf{k},\text{COMPRESS}}(n) \leq 9\log(n+1)\cdot\varepsilon_{\mathbf{k},\text{HALVE}}(\ell_n)$, a guarantee within $9\log(n+1)$ of the original KT(δ) MMD error (9). ■

Appendix F. Proof of Thm. 7: Runtime of COMPRESS++

First, the runtime bound (5) follows directly by adding the runtime of COMPRESS(HALVE, $\mathfrak{g}$) as given by (3) in Thm. 3 and the runtime of THIN.

Appendix G. Proof of Thm. 9: MMD guarantees for COMPRESS++

Noting that MMD is a metric, and applying triangle inequality, we have

$$\text{MMD}_{\mathbf{k}}(\mathcal{S}_{\text{in}}, \mathcal{S}_{\text{C++}}) \leq \text{MMD}_{\mathbf{k}}(\mathcal{S}_{\text{in}}, \mathcal{S}_{\text{C}}) + \text{MMD}_{\mathbf{k}}(\mathcal{S}_{\text{C}}, \mathcal{S}_{\text{C++}}).$$

Since $\mathcal{S}_{\text{C++}}$ is the output of THIN$(2^{\mathfrak{g}})$ with $\mathcal{S}_{\text{C}}$ as the input, applying the MMD tail bound assumption (24) with $|\mathcal{S}_{\text{C}}| = \sqrt{n}2^{\mathfrak{g}}$ substituted in place of n, we find that

$$\mathbb{P}\left[\text{MMD}(\mathcal{S}_{\text{C}}, \mathcal{S}_{\text{C++}}) \geq a'_{2^{\mathfrak{g}}\sqrt{n}} + v'_{2^{\mathfrak{g}}\sqrt{n}}\sqrt{t}\right] \leq e^{-t} \quad \text{for all} \quad t \geq 0.$$

Recall that $\ell_n/2 = 2^{\mathfrak{g}}\sqrt{n}$. Next, we apply Thm. 5 with HALVE to conclude that

$$\mathbb{P}[\text{MMD}_{\mathbf{k}}(\mathcal{S}_{\text{in}}, \mathcal{S}_{\text{C}}) \geq \widetilde{a}_n + \widetilde{v}_n \cdot \sqrt{t}] \leq e^{-t} \quad \text{for all} \quad t \geq 0.$$

Thus, we have

$$\mathbb{P}\left[\text{MMD}_{\mathbf{k}}(\mathcal{S}_{\text{in}}, \mathcal{S}_{\text{C++}}) \geq a'_{\ell_n/2} + \widetilde{a}_n + (v'_{\ell_n/2} + \widetilde{v}_n)\sqrt{t}\right] \leq 2\cdot e^{-t} \quad \text{for all} \quad t \geq 0,$$

which in turn implies that

$$\mathbb{P}\left[\text{MMD}_{\mathbf{k}}(\mathcal{S}_{\text{in}}, \mathcal{S}_{\text{C++}}) \geq a'_{\ell_n/2} + \widetilde{a}_n + (v'_{\ell_n/2} + \widetilde{v}_n)\sqrt{\log 2} + (v'_{\ell_n/2} + \widetilde{v}_n)\sqrt{t}\right] \leq e^{-t} \quad \text{for all} \quad t \geq 0,$$

thereby yielding the claimed result.

Appendix H. Examples for MMD for COMPRESS++

Example 4 (KT-COMPRESS++) In the notation of Ex. 2 and Thm. 20, consider running COMPRESS++ with THIN $=$ KT$(\frac{2^{\mathfrak{g}}-1}{\sqrt{n}-1}\delta)$ and HALVE $=$ symmetrized KT$(\frac{\ell}{2(n-\sqrt{n})}\delta)$ for HALVE inputs of size ℓ. As we detail in App. I.2, on an event of probability $1-\delta$, all COMPRESS++ invocations of HALVE and THIN are simultaneously $\mathbf{k}$-sub-Gaussian with parameters satisfying

$$\widetilde{\zeta}_{\mathrm{H}}(\ell_n) = \widetilde{\zeta}_{\mathrm{T}}(\tfrac{\ell_n}{2}) = C_v\sqrt{\|\mathbf{k}_{\mathrm{split}}\|_\infty \log(\tfrac{4(n-\sqrt{n})}{\delta})}\,\mathfrak{M}_{\mathcal{S}_{\mathrm{in}},\mathbf{k}_{\mathrm{split}}} \quad\Longrightarrow\quad \frac{\widetilde{\zeta}_{\mathrm{H}}(\ell_n)}{\widetilde{\zeta}_{\mathrm{T}}(\frac{\ell_n}{2})} = 1.$$

Since KT runs in $\Theta(n^2)$ time, Thms. 8 and 10 imply that KT-COMPRESS++ with $\mathfrak{g} = \lceil \log_2 \log n + 3\rceil$ runs in near-linear $\mathcal{O}(n\log^3 n)$ time and inflates $\mathbf{k}$-sub-Gaussian error by at most 4. ■

Appendix I. Proofs of Exs. 3 and 4

We begin by defining the notions of sub-Gaussianity and $\mathbf{k}$-sub-Gaussianity on an event.

Definition 21 (Sub-Gaussian on an event) *We say that a random variable G is* sub-Gaussian on an event $\mathcal{E}$ *with parameter σ if*

$$\mathbb{E}[\mathbf{1}[\mathcal{E}]\cdot\exp(\lambda\cdot G)] \le \exp(\tfrac{\lambda^2\sigma^2}{2}) \quad \textit{for all} \quad \lambda\in\mathbb{R}.$$

Definition 22 ($\mathbf{k}$-sub-Gaussian on an event) *For a kernel $\mathbf{k}$, we call a thinning algorithm* ALG $\mathbf{k}$-sub-Gaussian on an event $\mathcal{E}$ *with parameter v and shift a if*

$$\mathbb{P}[\mathcal{E}, \mathrm{MMD}_{\mathbf{k}}(\mathcal{S}_{\mathrm{in}},\mathcal{S}_{\mathrm{ALG}}) \ge a_n + v_n\sqrt{t} \mid \mathcal{S}_{\mathrm{in}}] \le e^{-t} \quad \textit{for all} \quad t\ge 0. \tag{24}$$

We will also make regular use of the unrolled representation for the COMPRESS measure discrepancy $\psi_{\mathrm{C}}(\mathcal{S}_{\mathrm{in}})$ in terms of the HALVE inputs $(\mathcal{S}^{\mathrm{in}}_{i,j})_{j\in[4^i]}$ of size $n_i = 2^{\mathfrak{g}+1-i}\sqrt{n}$ for $0\le i\le\beta_n$. For brevity, we will use the shorthand $\psi_{\mathrm{C}} \triangleq \psi_{\mathrm{C}}(\mathcal{S}_{\mathrm{in}})$, $\psi^{\mathrm{H}}_{i,j} \triangleq \psi_{\mathrm{H}}(\mathcal{S}^{\mathrm{in}}_{i,j})$, and $\psi_{\mathrm{T}} \triangleq \psi_{\mathrm{T}}(\mathcal{S}_{\mathrm{C}})$ hereafter.

I.1. Proof of Ex. 3: KT-COMPRESS

Since HALVE $=$ symmetrized KT$(\frac{\ell}{2n-\ell_n}\delta)$ when applied to an input of size ℓ, the proofs of Thms. 1 - 4 in Dwivedi and Mackey (2021a) identify a sequence of events $\mathcal{E}_{i,j}$ and random signed measures $\tilde{\psi}_{i,j}$ such that, for each $0\le i\le\beta_n$ and $j\in[4^i]$,

(a) $\mathbb{P}[\mathcal{E}^c_{i,j}] \le \frac{n_i}{2n-\ell_n}\delta$

(b) $\mathbf{1}[\mathcal{E}_{i,j}]\psi^{\mathrm{H}}_{i,j} = \mathbf{1}[\mathcal{E}_{i,j}]\tilde{\psi}_{i,j}$

(c) $\mathbb{P}[\frac{1}{n_i}\|\tilde{\psi}_{i,j}(\mathbf{k})\|_{\mathbf{k}} \ge a_{n_i} + v_{n_i}\sqrt{t} \mid (\tilde{\psi}_{i',j'})_{i'>i,j'\ge1}, (\tilde{\psi}_{i,j'})_{j'<j}] \le e^{-t}$ for all $t\ge0$

(d) $\mathbb{E}[\tilde{\psi}_{i,j}(\mathbf{k}) \mid (\tilde{\psi}_{i',j'})_{i'>i,j'\ge1}, (\tilde{\psi}_{i,j'})_{j'<j}] = 0.$

Hence, on the event $\mathcal{E} = \bigcap_{i,j}\mathcal{E}_{i,j}$, these properties hold simultaneously for all HALVE calls made by COMPRESS, and, by the union bound, $\mathbb{P}[\mathcal{E}^c]\le\delta$.

Furthermore, we may invoke the measure discrepancy representation, the equivalence of $\psi^{\mathrm{H}}_{i,j}$ and $\tilde{\psi}_{i,j}$ on $\mathcal{E}$, the nonnegativity of the exponential, and the proof of Thm. 5 in turn to find

$$\begin{aligned}
&\mathbb{P}[\mathcal{E}, \mathrm{MMD}(\mathcal{S}_{\mathrm{in}},\mathcal{S}_{\mathrm{C}}) \ge \tilde{a}_n + \tilde{v}_n\sqrt{t} \mid \mathcal{S}_{\mathrm{in}}] = \mathbb{P}[\mathcal{E}, \tfrac{1}{n}\|\psi_{\mathrm{C}}(\mathbf{k})\|_{\mathbf{k}} \ge \tilde{a}_n + \tilde{v}_n\sqrt{t} \mid \mathcal{S}_{\mathrm{in}}]\\
&= \mathbb{P}[\mathcal{E}, \tfrac{1}{n}\|\sqrt{n}2^{-\mathfrak{g}-1}\textstyle\sum_{i=0}^{\beta_n}\sum_{j=1}^{4^i}2^{-i}\tilde{\psi}_{i,j}(\mathbf{k})\|_{\mathbf{k}} \ge \tilde{a}_n + \tilde{v}_n\sqrt{t} \mid \mathcal{S}_{\mathrm{in}}]\\
&\le \mathbb{P}[\tfrac{1}{n}\|\sqrt{n}2^{-\mathfrak{g}-1}\textstyle\sum_{i=0}^{\beta_n}\sum_{j=1}^{4^i}2^{-i}\tilde{\psi}_{i,j}(\mathbf{k})\|_{\mathbf{k}} \ge \tilde{a}_n + \tilde{v}_n\sqrt{t} \mid \mathcal{S}_{\mathrm{in}}] \le e^{-t} \quad \text{for all} \quad t\ge0,
\end{aligned}$$

so that COMPRESS is $\mathbf{k}$-sub-Gaussian on $\mathcal{E}$ with parameters $(\tilde{v},\tilde{a})$.

I.2. Proof of Ex. 4: KT-COMPRESS++

In the notation of Ex. 2, define

$$\tfrac{\ell_n}{2} a_{\ell_n} = \sqrt{n} a'_{\ell_n/2} = C_a \sqrt{\|\mathbf{k}_{\text{split}}\|_\infty}, \quad \text{and}$$
$$\tfrac{\ell_n}{2} v_{\ell_n} = \sqrt{n} v'_{\ell_n/2} = C_v \sqrt{\|\mathbf{k}_{\text{split}}\|_\infty \log(\tfrac{4(n-\sqrt{n})}{\delta})}\, \mathfrak{M}_{\mathcal{S}_{\text{in}}, \mathbf{k}_{\text{split}}}.$$

Since THIN $=$ KT$(\frac{2^{\mathfrak{g}}-1}{\sqrt{n}-1}\delta)$ and HALVE $=$ symmetrized KT$(\frac{\ell}{2(n-\sqrt{n})}\delta)$ for HALVE inputs of size ℓ, the proofs of Thms. 1 - 4 in Dwivedi and Mackey (2021a) identify a sequence of events $\mathcal{E}_{i,j}$ and $\mathcal{E}_{\text{T}}$ and random signed measures $\tilde{\psi}_{i,j}$ and $\tilde{\psi}_{\text{T}}$ such that, for each $0 \le i \le \beta_n$ and $j \in [4^i]$,

(a) $\mathbb{P}[\mathcal{E}_{i,j}^c] \le \frac{n_i}{2(n-\sqrt{n})}\delta$ and $\mathbb{P}[\mathcal{E}_{\text{T}}^c] \le \frac{2^{\mathfrak{g}}-1}{\sqrt{n}-1}\delta$

(b) $\mathbf{1}[\mathcal{E}_{i,j}]\psi_{i,j}^{\text{H}} = \mathbf{1}[\mathcal{E}_{i,j}]\tilde{\psi}_{i,j}$ and $\mathbf{1}[\mathcal{E}_{\text{T}}]\psi_{\text{T}} = \mathbf{1}[\mathcal{E}_{\text{T}}]\tilde{\psi}_{\text{T}}$

(c) $\mathbb{P}[\frac{1}{n_i}\|\tilde{\psi}_{i,j}(\mathbf{k})\|_{\mathbf{k}} \ge a_{n_i} + v_{n_i}\sqrt{t} \mid (\tilde{\psi}_{i',j'})_{i'>i,j'\ge 1}, (\tilde{\psi}_{i,j'})_{j'<j}] \le e^{-t}$ and $\mathbb{P}[\frac{2}{\ell_n}\|\tilde{\psi}_{\text{T}}(\mathbf{k})\|_{\mathbf{k}} \ge a'_{\ell_n/2} + v'_{\ell_n/2}\sqrt{t} \mid \mathcal{S}_{\text{C}}] \le e^{-t}$ for all $t \ge 0$

(d) $\mathbb{E}[\tilde{\psi}_{i,j}(\mathbf{k}) \mid (\tilde{\psi}_{i',j'})_{i'>i,j'\ge 1}, (\tilde{\psi}_{i,j'})_{j'<j}] = 0.$

Hence, on the event $\mathcal{E} = \bigcap_{i,j} \mathcal{E}_{i,j} \cap \mathcal{E}_{\text{T}}$, these properties hold simultaneously for all HALVE calls made by COMPRESS and

$$\widetilde{\zeta}_{\text{H}}(\ell_n) = \widetilde{\zeta}_{\text{T}}(\tfrac{\ell_n}{2}) = C_v \sqrt{\|\mathbf{k}_{\text{split}}\|_\infty \log(\tfrac{4(n-\sqrt{n})}{\delta})}\, \mathfrak{M}_{\mathcal{S}_{\text{in}}, \mathbf{k}_{\text{split}}}.$$

Moreover, by the union bound, $\mathbb{P}[\mathcal{E}^c] \le \delta$.

Finally, since $\phi_{\text{C++}} = \frac{1}{n}(\psi_{\text{C}} + \psi_{\text{T}})$ and the argument of App. I.1 implies that COMPRESS is $\mathbf{k}$-sub-Gaussian on $\mathcal{E}$ with parameters $(\tilde{v}, \tilde{a})$, the triangle inequality implies that COMPRESS++ is $\mathbf{k}$-sub-Gaussian on $\mathcal{E}$ with parameters $(\hat{v}, \hat{a})$ as in App. G.

Appendix J. Experiments

We now turn to an empirical evaluation of the speed-ups and error of COMPRESS++.

J.1. Experiment set-up

We begin by describing the thinning algorithms, compression tasks, evaluation metrics, and kernels used in our experiments. Supplementary experimental details and results can be found in App. J.5.

Thinning algorithms Each experiment compares a high-accuracy, quadratic time thinning algorithm—either target kernel thinning (Dwivedi and Mackey, 2021a) or kernel herding (Chen et al., 2010)—with our near-linear time COMPRESS and COMPRESS++ variants that use the same input algorithm to HALVE and THIN. In each case, we perform root thinning, compressing n input points down to $\sqrt{n}$ points, so that COMPRESS is run with $\mathfrak{g} = 0$. For COMPRESS++, we use $\mathfrak{g} = 4$ throughout to satisfy the small relative error criterion (6) in all experiments. When halving we restrict each input algorithm to return distinct points and symmetrize the output as discussed in Thm. 20.

Compressing i.i.d. summaries To demonstrate the advantages of COMPRESS++ over equal-sized i.i.d. summaries we compress input point sequences $\mathcal{S}_{\text{in}}$ drawn i.i.d. from either (a) Gaussian targets $\mathbb{P} = \mathcal{N}(0, \mathbf{I}_d)$ with $d \in \{2, 4, 10, 100\}$ or (b) M-component mixture of Gaussian targets $\mathbb{P} = \frac{1}{M}\sum_{j=1}^{M} \mathcal{N}(\mu_j, \mathbf{I}_2)$ with $M \in \{4, 6, 8, 32\}$ and component means $\mu_j \in \mathbb{R}^2$ defined in App. J.5.

Compressing MCMC summaries To demonstrate the advantages of COMPRESS++ over standard MCMC thinning, we also compress input point sequences $\mathcal{S}_{\text{in}}$ generated by a variety of popular MCMC algorithms (denoted by RW, ADA-RW, MALA, and pMALA) targeting four challenging Bayesian posterior

distributions $\mathbb{P}$. In particular, we adopt the four posterior targets of Riabiz et al. (2020a) based on the *Goodwin (1965) model* of oscillatory enzymatic control ($d = 4$), the *Lotka (1925); Volterra (1926) model* of oscillatory predator-prey evolution ($d = 4$), the *Hinch et al. (2004) model* of calcium signalling in cardiac cells ($d = 38$), and a tempered Hinch model posterior ($d = 38$). Notably, for the Hinch experiments, each summary point discarded via an accurate thinning procedure saves 1000s of downstream CPU hours by avoiding an additional critically expensive whole-heart simulation (Riabiz et al., 2020a). More details on the MCMC algorithms and targets can be found in App. J.5.

Kernel settings Throughout we use a Gaussian kernel $\mathbf{k}(x, y) = \exp(-\frac{1}{2\sigma^2}\|x - y\|_2^2)$ with σ^2 as specified by Dwivedi and Mackey (2021b, Sec. K.2) for the MCMC targets and $\sigma^2 = 2d$ otherwise.

Evaluation metrics For each thinning procedure we report mean runtime across 3 runs and mean MMD error across 10 independent runs $\pm$ 1 standard error (the error bars are often too small to be visible). All runtimes were measured on a single core of an Intel Xeon CPU with 32GB RAM. For the i.i.d. targets, we report $\mathrm{MMD}_{\mathbf{k}}(\mathbb{P}, \mathbb{P}_{\mathrm{out}})$ which can be exactly computed in closed-form. For the MCMC targets, we report the thinning error $\mathrm{MMD}_{\mathbf{k}}(\mathbb{P}_{\mathrm{in}}, \mathbb{P}_{\mathrm{out}})$ analyzed directly by our theory (Thms. 5 and 9). We also display an empirical rate of decay on each MMD plot.

J.2. Kernel thinning results

We first apply COMPRESS++ to the near-optimal KT algorithm to obtain comparable summaries at a fraction of the cost. Figs. 3 and 4 reveal that, in line with our guarantees, KT-COMPRESS++ matches or nearly matches the MMD error of KT in all experiments while also substantially reducing runtime. For example, KT thins $65,000$ points in 10 dimensions in 20m, while KT-COMPRESS++ needs only 1.5m; KT takes more than a day to thin $250,000$ points in 100 dimensions, while KT-COMPRESS++ takes less than an hour (a $32\times$ speed-up). For reference we also display the error of standard thinning (ST) to highlight that KT-COMPRESS++ significantly improves approximation quality relative to the standard practice of i.i.d. summarization or standard MCMC thinning. See Fig. 7 in App. J.6 for analogous results with mixture of Gaussian targets.

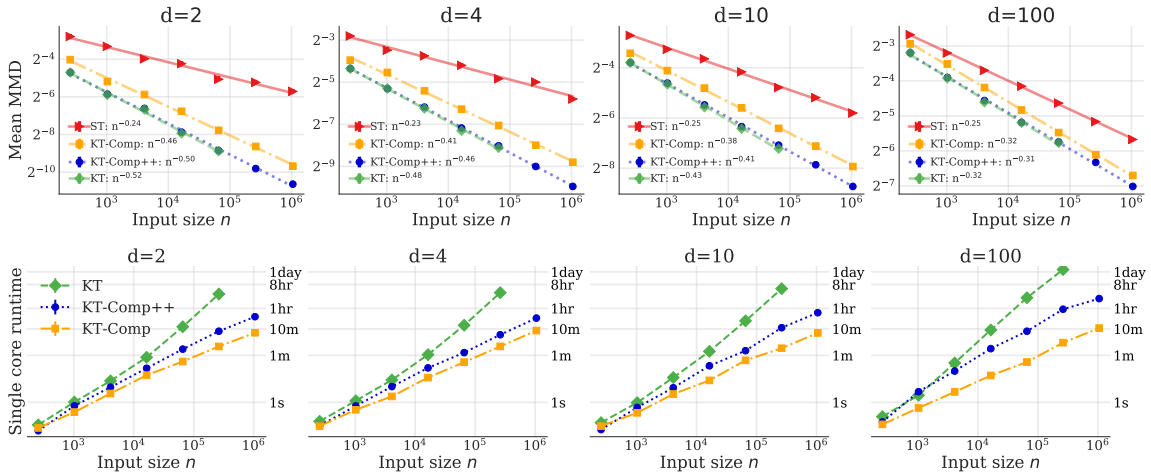

Figure 3: For Gaussian targets $\mathbb{P}$ with $d \in \{2, 4, 10, 100\}$, KT-COMPRESS++ improves upon the MMD of i.i.d. sampling (ST), closely tracks the error of its quadratic-time input algorithm KT, and substantially reduces the runtime. See App. J.2 for more details.

J.3. Kernel herding results

A strength of COMPRESS++ is that it can be applied to any thinning algorithm, including those with suboptimal or unknown performance guarantees that often perform well in practical. In such cases, Thm. 6 still

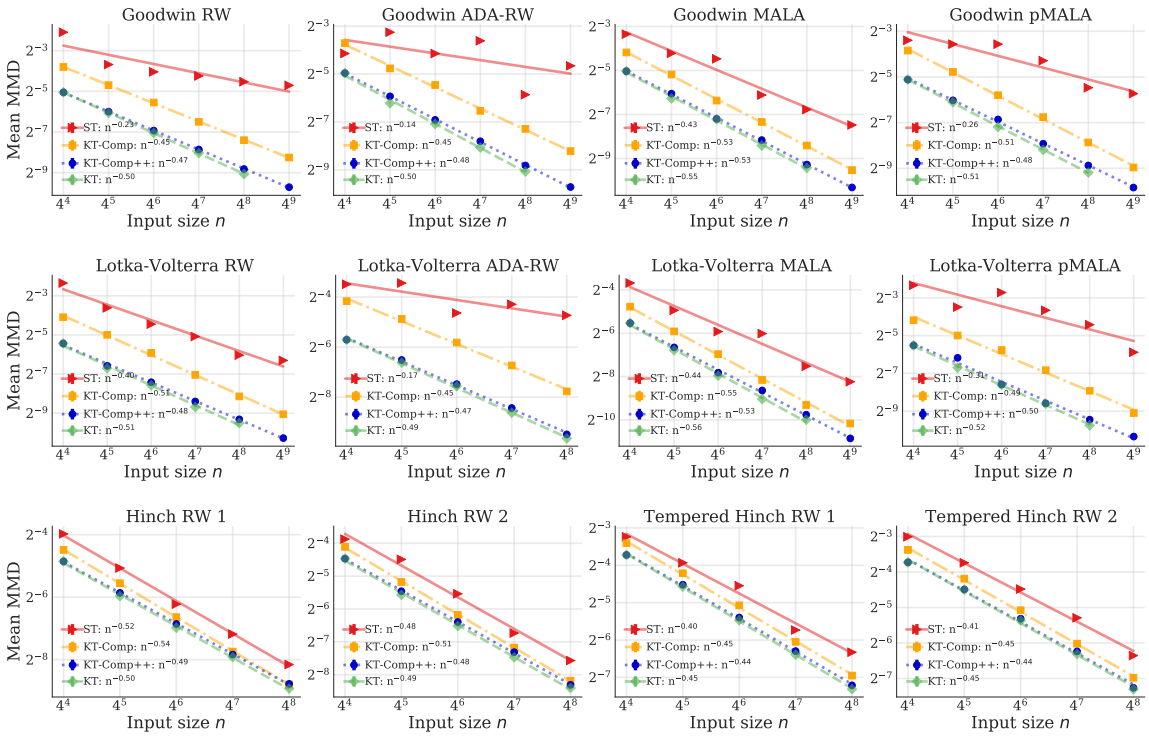

Figure 4: Given MCMC sequences summarizing challenging differential equation posteriors $\mathbb{P}$, KT-COMPRESS++ consistently improves upon the MMD of standard thinning (ST) and matches or nearly matches the error of of its quadratic-time input algorithm KT. See App. J.2 for more details.

ensures that COMPRESS++ error is never much larger than that of the input algorithm. As an illustration, we apply COMPRESS++ to the popular quadratic-time kernel herding algorithm (Herd). Fig. 5 shows that Herd-COMPRESS++ matches or nearly matches the MMD error of Herd in all experiments while also substantially reducing runtime. For example, Herd requires more than 11 hours to compress $250,000$ points in 100 dimensions, while Herd-COMPRESS++ takes only 14 minutes (a $45\times$ speed-up). Moreover, surprisingly, Herd-COMPRESS++ is consistently more accurate than the original kernel herding algorithm for lower dimensional problems. See Fig. 7 in App. J.6 for comparable results with mixture of Gaussian targets.

J.4. Visualizing coresets

For a 32-component mixture of Gaussians target, Fig. 6 visualizes the coresets produced by i.i.d. sampling, KT, kernel herding, and their COMPRESS++ variants. The COMPRESS++ coresets closely resemble those of their input algorithms and, compared with i.i.d. sampling, yield visibly improved stratification across the mixture components.

J.5. Supplementary Details for Experiments

In this section, we provide supplementary experiment details deferred from Sec. 5, as well as some additional results.

In all experiments involving kernel thinning, we set the algorithm failure probability parameter $\delta = \frac{1}{2}\frac{\sqrt{n}-1}{\sqrt{n}}$ and compare $\mathrm{KT}(\delta)$ to COMPRESS and COMPRESS++ with HALVE and THIN set as in Exs. 3 and 4 respectively.

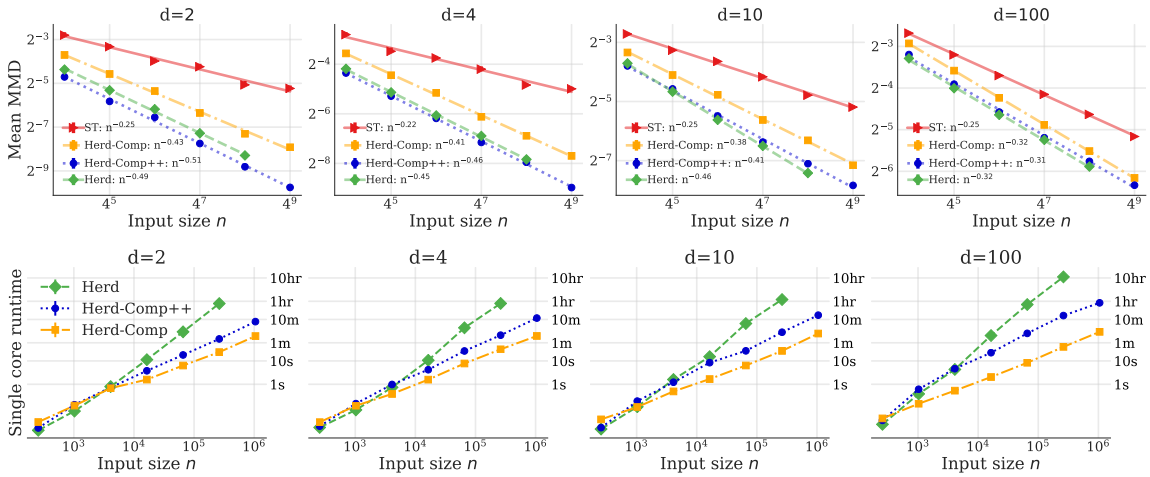

Figure 5: Herd-COMPRESS++ improves upon the MMD of i.i.d. sampling (ST), closely tracks or improves upon the error of its quadratic-time input algorithm kernel herding (Herd), and substantially reduces the runtime. See App. J.3 for more details.

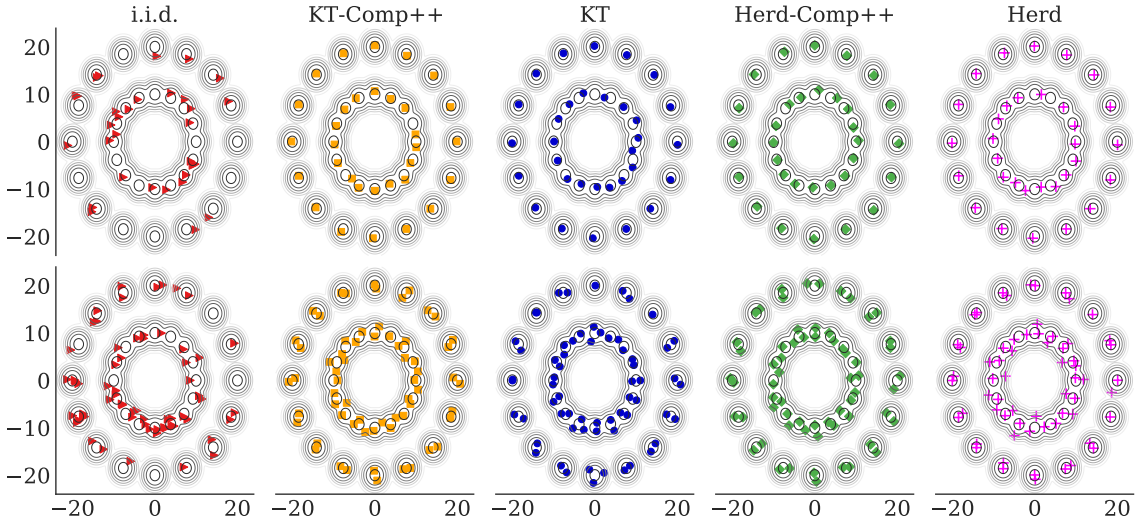

Figure 6: Coresets of size 32 (top) or 64 (bottom) for 32-component mixture of Gaussian target with equidensity contours of the target underlaid. See App. J.4 for more details.

J.6. Mixture of Gaussian target details and MMD plots

For the target used for coreset visualization in Fig. 6, the mean locations are on two concentric circles of radii 10 and 20, and are given by

$$\mu_j = \alpha_j \begin{bmatrix} \sin(j) \\ \cos(j) \end{bmatrix} \quad \text{where } \alpha_j = 10 \cdot \mathbf{1}(j \leq 16) + 20 \cdot \mathbf{1}(j > 16) \qquad \text{for } j = 1, 2, \ldots, 32.$$

Here we also provide additional results with mixture of Gaussian targets given by $\mathbb{P} = \frac{1}{M}\sum_{j=1}^{M}\mathcal{N}(\mu_j, \mathbf{I}_d)$ for $M \in \{4, 6, 8\}$. The mean locations for these are given by

$$\mu_1 = [-3, 3]^\top, \quad \mu_2 = [-3, 3]^\top, \quad \mu_3 = [-3, -3]^\top, \quad \mu_4 = [3, -3]^\top,$$
$$\mu_5 = [0, 6]^\top, \quad \mu_6 = [-6, 0]^\top, \quad \mu_7 = [6, 0]^\top, \quad \mu_8 = [0, -6]^\top.$$

Fig. 7 plots the MMD errors of KT and herding experiments for the mixture of Gaussians targets with $4, 6$ and 8 centers, and notice again that COMPRESS++ provides a competitive performance to the original algorithm, in fact suprisingly, improves upon herding.

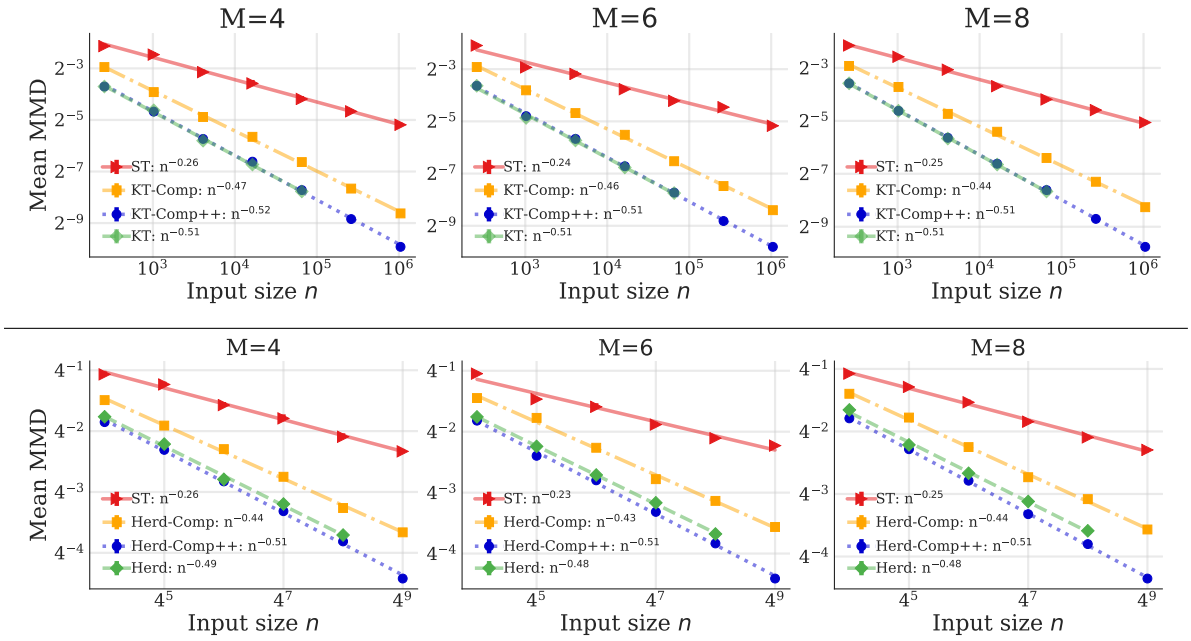

Figure 7: For M-component mixture of Gaussian targets, KT-COMPRESS++ and Herd-COMPRESS++ improve upon the MMD of i.i.d. sampling (ST) and closely track or improve upon the error of their quadratic-time input algorithms, KT and kernel herding (Herd). See App. J.6 for more details.

J.7. Details Of MCMC Targets

Our set-up for the MCMC experiments is identical to that of Dwivedi and Mackey (2021b, Sec. 6), except that we use all post-burn-in points to generate our Goodwin and Lotka-Volterra input point sequences $\mathcal{S}_{\text{in}}$ instead of only the odd indices. In particular, we use the MCMC output of Riabiz et al. (2020b) described in (Riabiz et al., 2020a, Sec. 4) and perform thinning experiments after discarding the burn-in points. To generate an input $\mathcal{S}_{\text{in}}$ of size n for a thinning algorithm, we downsample the post-burn-in points using standard thinning. For Hinch, we additionally do coordinate-wise normalization by subtracting the sample mean and dividing by sample standard deviation of the post-burn-in-points.

In Sec. 5, RW and ADA-RW respectively refer to Gaussian random walk and adaptive Gaussian random walk Metropolis algorithms (Haario et al., 1999) and MALA and pMALA respectively refer to the Metropolis-adjusted Langevin algorithm (Roberts and Tweedie, 1996) and pre-conditioned MALA (Girolami and Calderhead, 2011). For Hinch experiments, RW 1 and RW 2 refer to two independent runs of Gaussian random walk, and "Tempered" denotes the runs targeting a tempered Hinch posterior. For more details on the set-up, we refer the reader to Dwivedi and Mackey (2021b, Sec. 6.3, App. J.2).

Appendix K. Streaming Version of COMPRESS

COMPRESS can be efficiently implemented in a streaming fashion (Alg. 3) by viewing the recursive steps in Alg. 1 as different levels of processing, with the bottom level denoting the input points and the top level denoting the output points. The streaming variant of the algorithm efficiently maintains memory at several levels and processes inputs in batches of size $4^{\mathfrak{g}+1}$. At any level i (with $i = 0$ denoting the level of the input points), whenever there are $2^i 4^{\mathfrak{g}+1}$ points, the algorithm runs HALVE on the points in this level, appends the output of size $2^{i-1}4^{\mathfrak{g}+1}$ to the points at level $i+1$, and empties the memory at level i (and thereby level i never stores more than $2^i 4^{\mathfrak{g}+1}$ points). In this fashion, just after processing $n = 4^{k+\mathfrak{g}+1}$ points, the highest level is $k+1$, which contains a compressed coreset of size $2^{k-1}4^{\mathfrak{g}+1} = 2^{k+\mathfrak{g}+1}2^{\mathfrak{g}} = \sqrt{n}2^{\mathfrak{g}}$ (outputted by running HALVE at level k for the first time), which is the desired size for the output of COMPRESS.

Algorithm 3: COMPRESS (Streaming) – Outputs stream of coresets of size $2^{\mathfrak{g}}\sqrt{n}$ for $n = 4^{k+\mathfrak{g}+1}$ and $k \in \mathbb{N}$

Input: halving algorithm HALVE, oversampling parameter $\mathfrak{g}$, stream of input points $x_1, x_2, \ldots$

```
S_0 ← {}                                          // Initialize empty level 0 coreset
for t = 1, 2, ..., do
    S_0 ← S_0 ∪ (x_j)_{j=1+(t-1)·4^{g+1}}^{t·4^{g+1}}   // Process input in batches of size 4^{g+1}
    if t == 4^j for j ∈ N then
        S_{j+1} ← {}                              // Initialize level j+1 coreset after processing 4^{j+g+1} input points
    end
    for i = 0, ..., ⌈log_4 t⌉ + 1 do
        if |S_i| == 2^i 4^{g+1} then
            S ← HALVE(S_i)                        // Halve level i coreset to size 2^{i-1}4^{g+1}
            S_{i+1} ← S_{i+1} ∪ S                 // Update level i+1 coreset: has size ∈ {1,2,3,4}·2^{i-1}4^{g+1}
            S_i ← {}                              // Empty coreset at level i
        end
    end
    if t == 4^j for j ∈ N then
        output S_{j+1}                            // Coreset of size √n 2^g with n ≜ t4^{g+1} and t = 4^j for j ∈ N
    end
end
```

Our next result analyzes the space complexity of the streaming variant (Alg. 3) of COMPRESS. The intuition for gains in memory requirements is very similar to that for running time, as we now maintain (and run HALVE) on subsets of points with size much smaller than the input sequence. We count the number of data points stored as our measure of memory.

Proposition 23 (COMPRESS Streaming Memory Bound) *Let* HALVE *store $s_{\mathrm{H}}(n)$ data points on inputs of size n. Then, after completing iteration t, the streaming implementation of* COMPRESS *(Alg. 3) has used at most $s_{\mathrm{C}}(t) = 4^{\mathfrak{g}+3}\sqrt{t} + s_{\mathrm{H}}(2^{\mathfrak{g}+1}\sqrt{t})$ data points of memory.*

Proof At time t, we would like to estimate the space usage of the algorithm. At the ith level of memory, we can have at most $2^{i+2}4^{\mathfrak{g}}$ data points. Since we are maintaining a data set of size at most $\sqrt{t}4^{\mathfrak{g}}$ at time t, there are at most $\frac{\log t}{2}$ levels. Thus, the maximum number of points stored at time t is bounded by

$$\sum_{i=0}^{0.5\log t} 2^{i+2}4^{\mathfrak{g}} \leq 4^{\mathfrak{g}+3}\sqrt{t}.$$

Furthermore, at any time up to time t, we have run HALVE on a point sequence of size at most $\sqrt{t}2^{\mathfrak{g}+1}$ which requires storing at most $s_{\mathrm{H}}(\sqrt{t}2^{\mathfrak{g}+1})$ additional points. ■

Example 5 (KT-COMPRESS and KT-COMPRESS++) First consider the streaming variant of COMPRESS with HALVE = symmetrized KT($\frac{\ell}{2n-\ell_n}\delta$) for HALVE inputs of size ℓ as in Ex. 3. Since $s_{\mathrm{KT}}(n) \leq n$ (Dwivedi and Mackey, 2021b, Sec. 3), Thm. 23 implies that $s_{\mathrm{C}}(n) \leq 4^{\mathfrak{g}+4}\sqrt{n}$.

Next consider COMPRESS++ with the streaming variant of COMPRESS, THIN $=$ KT($\frac{2^{\mathfrak{g}}-1}{\sqrt{n}-1}\delta$), and HALVE $=$ symmetrized KT($\frac{\ell}{2(n-\sqrt{n})}\delta$) for HALVE inputs of size ℓ as in Ex. 4. The space complexity $s_{\mathrm{C++}}(n) = s_{\mathrm{C}}(n) + s_{\mathrm{KT}}(\ell_n) = 4^{\mathfrak{g}+4}\sqrt{n} + \ell_n \leq 4^{\mathfrak{g}+5}\sqrt{n}$. Setting $\mathfrak{g}$ as in Ex. 4, we get $s_{\mathrm{C++}}(n) = \mathcal{O}(\sqrt{n}\log^2 n)$. ■