DISTRIBUTION COMPRESSION IN NEAR-LINEAR TIME

Anonymous authors
Paper under double-blind review

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 $\widetilde{\mathcal{O}}(1/\sqrt{n})$ distributional discrepancy to $\mathbb{P}$. Unfortunately, these same algorithms suffer from quadratic or super-quadratic runtime in the sample size n. To address this deficiency, we introduce a simple meta-procedure—Compress++—for speeding up any input thinning algorithm while suffering at most a factor of four 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 $\mathcal{O}(\sqrt{\log n/n})$ integration error and better-than-Monte-Carlo maximum mean discrepancy in $\mathcal{O}(n \log^2 n)$ 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 long-running 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 & 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 $\mathbb{P}_n f \triangleq \frac{1}{n}\sum_{i=1}^n f(x_i)$ 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 $|\mathbb{P}_n f - \mathbb{P}f|$, 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), or expert data annotation which can be monetarily expensive Beaugnon et al. (2017), and is sometimes coined as the human bottleneck of machine learning.

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 & Mackey (2021b;a) introduced a more effective alternative, *kernel thinning*, that provides near optimal $\widetilde{\mathcal{O}}(n^{-\frac{1}{2}})$ error when compressing n points 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.

Problem setup 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 $|\mathbb{P}f - \mathbb{P}_{\text{out}}f| \triangleq |\mathbb{P}f - \frac{1}{|\mathcal{S}_{\text{out}}|}\sum_{x\in\mathcal{S}_{\text{out}}} f(x)|$ for functions f in the reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ induced by a given kernel $\mathbf{k}$ (Berlinet & 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:

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

Our contributions 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 $\mathcal{O}(n^2)$ runtime to near linear $\mathcal{O}(n \log^2 n)$ time while maintaining the error guarantees up to a factor four. Since the $\widetilde{\mathcal{O}}(n^{-\frac{1}{2}})$ KT MMD guarantees of Dwivedi & Mackey (2021b) match the $\Omega(n^{-\frac{1}{2}})$ minimax lower bounds of Tolstikhin et al. (2017); Phillips & 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).

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), support points (Mak & Joseph, 2018), Stein points (MCMC) (Chen et al., 2018; 2019), and Stein thinning (Riabiz et al., 2020a), all of which run in $\Omega(n^2)$ time. As a demonstration, we combine COMPRESS++ with the popular kernel herding algorithm and observe 10-60$\times$ speed-ups. In all of our experiments, COMPRESS++ leads to minimal loss in accuracy and, surprisingly, even improves upon herding accuracy for lower-dimensional problems.

Overview of COMPRESS++ To define COMPRESS++, we first introduce COMPRESS: A simple and elegant recursive strategy takes in a halving algorithm and provides an intermediate thinned coreset in significantly faster runtime. COMPRESS divides the coreset into four parts, recursively applies COMPRESS to each, combines the output of each and halves the resulting coreset to get the output. Then COMPRESS++ with a thinning algorithm ALG is defined as follows: In stage one, use COMPRESS with ALG instantiated for 2-thinning, to obtain an intermediate coreset of size $2^{\mathfrak{g}}\sqrt{n}$ for a suitable parameter $\mathfrak{g}$, and then in stage two apply ALG directly to further thin it down to $\sqrt{n}$ points. In this manner, ALG is ever applied to coresets of size $2^{\mathfrak{g}}\sqrt{n}$ or smaller thereby improving the runtime. The parameter $\mathfrak{g}$ is chosen based on the run-time and known guarantees of the underlying thinning algorithm, but a default choice of 4 can be shown to be optimal in several theoretical settings, and it provided competitive performance across all our experiments. Overall, with n input points and a thinning algorithm with runtime n^{α}, COMPRESS++ uses a outputs $\sqrt{n}$ points in time $\widetilde{\mathcal{O}}(n^{\alpha/2})$ with errors similar to those provided by the input thinning algorithm.

Notation We write $\mathcal{S}_{\text{ALG}}$ for the coreset outputted by an algorithm ALG and extend our MMD definition for point sequences $(\mathcal{S}_1, \mathcal{S}_2)$ with empirical distributions $(\mathbb{Q}_1, \mathbb{Q}_2)$ via $\text{MMD}_{\mathbf{k}}(\mathcal{S}_1, \mathcal{S}_2) \triangleq \text{MMD}_{\mathbf{k}}(\mathbb{Q}_1, \mathbb{Q}_2)$ and $\text{MMD}_{\mathbf{k}}(\mathbb{P}, \mathcal{S}_1) \triangleq \text{MMD}_{\mathbf{k}}(\mathbb{P}, \mathbb{Q}_1)$. We use $a \precsim b$ and $a \succsim b$ to mean $a = \mathcal{O}(b)$ and $a = \Omega(b)$ and use $\precsim_{\mathbf{k}}$ to denote that the underlying constants depend on $\mathbf{k}$.

Basic definitions We use $\mathcal{H}$ to denote an inner-product space endowed with the inner-product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$. We start with the definition of sub-gamma random variables (Boucheron et al., 2013).

Definition 1 (f-Sub-Gamma) *A random variable X is said to be sub-gamma on the right with parameters (σ^2, c) denoted by $\Gamma_+(\sigma^2, c)$ if for all $0 < \lambda < \frac{1}{c}$, we have $\mathbb{E}[\exp(\lambda X)] \leq \exp\left(\frac{\lambda^2\sigma^2}{2(1-c\lambda)}\right)$. For any $f \in \mathcal{H}$, we say that a random variable G on $\mathcal{H}$ is f-sub-gamma on the right, denoted by $\Gamma_+^f(\sigma^2, c)$, if the random variable $\langle f, G\rangle_{\mathcal{H}} \in \Gamma_+(\sigma^2, c)$.*

As a reminder, we note that $c = 0$ yields a sub-Gaussian tails on the right. For $X \in \Gamma_+(\sigma^2, c)$, Boucheron et al. (2013, Section 2.4) shows that for $t \geq 0$, and $\delta \in (0, 1]$, we have

$$\mathbb{P}\big[X > \sqrt{2\sigma^2 t} + ct\big] \leq e^{-t}, \quad \text{or equivalently} \quad \mathbb{P}\big[X \leq \sqrt{2\sigma^2 \log(\tfrac{1}{\delta})} + c\log(\tfrac{1}{\delta})\big] \geq 1 - \delta. \quad (1)$$

A property that we frequently use is that the sub-gamma property is suitably closed under multiplication and addition of sub-gamma random variables. A discussion is deferred to App. A.

Definition 2 (Thinning and halving algorithms) *Consider an algorithm* ALG *that takes as input a point sequence $\mathcal{S}_{\text{in}}$ of size n and returns a (possibly random) subsequence $\mathcal{S}_{\text{ALG}} \subset \mathcal{S}_{\text{in}}$ of size n_{out}. We say that* ALG *is α_n-thinning if $n_{\text{out}} = n/\alpha_n$ whenever $n/\alpha_n \in \mathbb{N}$. When $\alpha_n = 2$, we say that* ALG *is a Halving algorithm, and when $\alpha_n = \sqrt{n}$, we say* ALG *is root-thinning.*

For a thinning algorithm ALG, we associate a kernel discrepancy embedding that measures the approximation quality for the n input points $\mathcal{S}_{\text{in}}$ provided by the n_{out} points $\mathcal{S}_{\text{ALG}}$ output by ALG:

$$\psi_{\text{ALG}}(\mathcal{S}_{\text{in}}) \triangleq \sum_{x \in \mathcal{S}_{\text{in}}} \mathbf{k}(x, \cdot) - \frac{n}{n_{\text{out}}} \sum_{x \in \mathcal{S}_{\text{ALG}}} \mathbf{k}(x, \cdot). \tag{2}$$

Letting $\mathbb{P}_{\mathcal{S}}$ denote the empirical distribution of $\mathcal{S}$, the reproducing property of $\mathbf{k}$ yieldsthat

$$\mathbb{P}_{\mathcal{S}_{\text{in}}} f - \mathbb{P}_{\mathcal{S}_{\text{ALG}}} f = \frac{1}{n} \langle f, \psi_{\text{ALG}} \rangle_{\mathbf{k}}, \text{ for any } f \in \mathcal{H}, \quad \text{and} \quad \text{MMD}(\mathcal{S}_{\text{in}}, \mathcal{S}_{\text{ALG}}) = \frac{1}{n} \|\psi_{\text{ALG}}\|_{\mathbf{k}}. \tag{3}$$

Next, we define the notion of a sub-gamma thinning algorithm via the object ψ_{ALG} (2).

Definition 3 (Sub-Gamma Thinning Algorithm) *Given the functions $f \in \mathcal{H}$, $\sigma : \mathbb{N} \to \mathbb{R}_+$ and $c : \mathbb{N} \to \mathbb{R}_+$, an α_n-thinning algorithm* ALG *is called f-sub-gamma on the right with parameters (σ^2, c), if $\psi_{\text{ALG}}(\mathcal{S}_{\text{in}}) \in \Gamma_+^f(\sigma^2(n), c(n))$ conditioned on $\mathcal{S}_{\text{in}}$ of size n.*

When the algorithm is both Γ_+^f and Γ_+^{-f}-sub-gamma, (1) and (3) immmediately imply a high probability tail bound on the integration error $|\mathbb{P}_{\mathcal{S}_{\text{in}}} f - \mathbb{P}_{\mathcal{S}_{\text{ALG}}} f|$.

2 COMPRESS

We now introduce our first meta procedure COMPRESS (which is also a building block for COMPRESS++). COMPRESS takes as input a halving algorithm HALVE and a oversampling factor $\mathfrak{g}$, and then given any input of size n, it outputs at thinned coreset of size $2^{\mathfrak{g}}\sqrt{n}$.

The algorithm is extremely simple to implement: first, divide the input sequence 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 $2^{\mathfrak{g}}\sqrt{n}$. We denote this algorithm as COMPRESS(HALVE, $\mathfrak{g}$), and describe it formally in Alg. 1. COMPRESS can also be implemented in a streaming fashion but we defer the discussion to App. G. We remind that COMPRESS with $\mathfrak{g} = 0$ is a root thinning algorithm.

Algorithm 1: COMPRESS

Input: Halving algorithm HALVE, oversampling factor $\mathfrak{g}$, 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 subsets $\{\mathcal{S}_i\}_{i=1}^4$ each of size $n/4$
  for $i = 1, 2, 3, 4$ **do**
    $\widetilde{\mathcal{S}}_i \leftarrow$ COMPRESS($\mathcal{S}_i$, 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 $2 \cdot 2^{\mathfrak{g}} \cdot \sqrt{n}$
  $\mathcal{S} \leftarrow$ HALVE $(\mathbf{k}, \widetilde{\mathcal{S}})$ // coreset of size $2^{\mathfrak{g}}\sqrt{n}$
  return $\mathcal{S}$
end

2.1 SUB-GAMMA PROPERTY AND RUNTIME GUARANTEES FOR COMPRESS

We now establish the sub-gamma property and runtime of COMPRESS in terms of the input halving algorithm HALVE. We measure computational complexity in terms of the dominant operations done by the halving algorithms (e.g., kernel computations with kernel thinning), and ignore the time taken for reading and writing to memory. See App. B for the proof. In the result, our notation highlights explicitly the dependence of σ^2, c and r_{CP} on $\mathfrak{g}$.

Theorem 1 (Sub-gamma property and runtime of COMPRESS) *Let* HALVE *denote a* Γ_+^f*-halving algorithm with parameters* $(\sigma^2_{\text{HALVE}}, c_{\text{HALVE}})$*, with running time* $r_{\text{HALVE}}(n)$ *on inputs of size* n*. Then with inputs of size* n*,* COMPRESS(HALVE, $\mathfrak{g}$) *outputs a coreset of size* $2^{\mathfrak{g}}\sqrt{n}$*, and is* Γ_+^f*-thinning with parameters*

$$\sigma^2_{\text{CP}}(n,\mathfrak{g}) \triangleq \tfrac{n}{4^{\mathfrak{g}+1}} \textstyle\sum_{i=0}^{\log_4 n-\mathfrak{g}-1} \sigma^2_{\text{HALVE}}\big(2^{\mathfrak{g}+1-i}\sqrt{n}\big) \leq \tfrac{n}{4^{\mathfrak{g}+1}}(\log_4 n-\mathfrak{g})\sigma^2_{\text{HALVE}}\big(2^{\mathfrak{g}+1}\sqrt{n}\big), \quad (4)$$
$$c_{\text{CP}}(n,\mathfrak{g}) \triangleq \max_{0\leq i\leq \log_4 n-\mathfrak{g}-1}\Big[\tfrac{\sqrt{n}}{2^{\mathfrak{g}+i+1}} c_{\text{HALVE}}\big(\sqrt{n}2^{\mathfrak{g}+1-i}\big)\Big],$$

and has the computational complexity bounded by

$$r_{\text{CP}}(n,\mathfrak{g}) = \textstyle\sum_{i=0}^{\log_4 n-\mathfrak{g}-1} 4^i r_{\text{HALVE}}(2^{\mathfrak{g}+1}\sqrt{n4^{-i}}).$$

Remark 1 (Worst-case guarantee for COMPRESS) For any monotone σ^2_{HALVE} and c_{HALVE}, we have $\frac{\sigma^2_{\text{CP}}(n,0)}{n^2} \leq \frac{1}{4n}\sigma^2_{\text{HALVE}}\big(2\sqrt{n}\big)(\log_4 n)$ and $\frac{c_{\text{CP}}}{n} \leq \frac{1}{2\sqrt{n}} c_{\text{HALVE}}\big(2\sqrt{n}\big)$. Recall from (1), this gives us a high probability bound on $\frac{1}{n}\langle f, \psi_{\text{CP}}\rangle_{\mathcal{H}}$. From (3), this bounds the integration error $\mathbb{P}_{\mathcal{S}_{\text{in}}} f - \mathbb{P}_{\mathcal{S}_{\text{CP}}} f$. Comparing this with the corresponding expressions for HALVE for compressing an input of size $2\sqrt{n}$ to $\sqrt{n}$, we find that in the worst case the error from COMPRESS is only a logarithmic factor larger. In particular, this shows that the COMPRESS algorithm gives guarantees that are only a logarithmic factor increase in error compared other methods of combining HALVE to get a root-thinning algorithm (such as RECHALVE discussed in App. H).

Remark 2 (Explicit speed-up of runtimes) Let us demonstrate how COMPRESS speeds up a thinning algorithm. For ease of discussion set $\mathfrak{g} = 0$. First, we note that $r_{\text{CP}}(n) \lesssim \log n \cdot r_{\text{HALVE}}\big(2\sqrt{n}\big)$. Thus, the running time is upper bounded by a logarithmic factor over the running HALVE on *a much smaller input of size* $2\sqrt{n}$. Consequently, COMPRESS provides a significant improvement for any halving algorithm with super-linear runtime. Unrolling the recursion for explicit polynomial runtime, we find that

- $r_{\text{HALVE}}(n) = Cn^2 \quad \Longrightarrow \quad r_{\text{CP}}(n) = C(\log n - \mathfrak{g})4^{\mathfrak{g}+1}n$
- $r_{\text{HALVE}}(n) = Cn^k$ for $k > 2 \quad \Longrightarrow \quad r_{\text{CP}} = Cn^{k/2}2^{k\mathfrak{g}+k}$

Note that in both these setting, we get an approximately quadratic speed up. In particular, for kernel thinning the speed up is from $\mathcal{O}(n^2)$ to $\mathcal{O}(n\log n)$, which also provides significant speed up in practice as our experiments later demonstrate.

Remark 3 (Explicit bounds on sub-gamma parameters) Unrolling the recursion (4), we now derive a explicit expressions for $\sigma^2_{\text{CP}}(n,\mathfrak{g})$ under the following growth conditions on σ^2_{HALVE}:

- $\sigma^2_{\text{HALVE}}(n) = C\log_4 n \quad \Longrightarrow \quad \sigma^2_{\text{CP}}(n,\mathfrak{g}) = C\frac{n}{4^{\mathfrak{g}+1}} \cdot \frac{1}{4}(\log_4 n + 3\mathfrak{g} + 3)(\log_4 n - \mathfrak{g})$
- $\sigma^2_{\text{HALVE}}(n) = C \quad \Longrightarrow \quad \sigma^2_{\text{CP}}(n,\mathfrak{g}) = Cn4^{-\mathfrak{g}-1}(\log_4 n - \mathfrak{g})$.

Example 1 (KT-SPLIT meets COMPRESS.) We now illustrate the gains of COMPRESS with the KT-SPLIT algorithm introduced in Dwivedi & Mackey (2021a, Alg1a.). Given a target kernel $\mathbf{k}$, KT-SPLIT uses non-uniform randomness to recursively thin an input sequence $\mathcal{S}_{\text{in}}$ of size n to a sequence $\mathcal{S}_{\text{out}}$ of size n/α. Its runtime is bounded as $\mathcal{O}(n^2)$, and thus Rem. 2 immediately yields that COMPRESS leads to an $\mathcal{O}(\frac{n}{\log n})$ speed-up in runtime—thereby reducing the runtime of KT from days to 10 minutes for a million samples in 100 dimensions as seen in Fig. 1.

Next, we show that the single function error of COMPRESS is at most $\sqrt{\log n}$ times worse than that of KT-SPLIT. With $\alpha = 2$, Dwivedi & Mackey (2021a, Thm 1, Rem. 1a) then implies that for any f in the RKHS of $\mathbf{k}$, we have

$$\tfrac{1}{n}\Big|\textstyle\sum_{x\in\mathcal{S}_{\text{in}}} f(x) - 2\sum_{x\in\mathcal{S}_{\text{out}}} f(x)\Big| \leq \tfrac{1}{n}\sqrt{\tfrac{32\log 4}{3}\|\mathbf{k}\|_\infty}\|f\|_{\mathbf{k}}\Big(\sqrt{\log(2n)\log\big(\tfrac{1}{\delta}\big)} + \log\big(\tfrac{1}{\delta}\big)\Big)$$

with probability $1-\delta$. Inverting this bound, we obtain that $\Pr\left[\left|\sum_{x\in\mathcal{S}_{\rm in}} f(x) - 2\sum_{x\in\mathcal{S}_{\rm out}} f(x)\right| \geq c_n t + \sqrt{2\sigma_n^2 t}\right] \leq e^{-t}$ for $c_n = \sqrt{\frac{32\log 4}{3}\|\mathbf{k}\|_\infty}\|f\|_{\mathbf{k}}$ and $\sigma_n^2 = \frac{4\log 4}{3}\log(2n)\cdot\|f\|_{\mathbf{k}}^2\|\mathbf{k}\|_\infty$. Applying (Boucheron et al., 2013, Theorem 2.3), we thus conclude that KT-SPLIT (in the halving mode) is $\Gamma_+^f\big(32\sigma_n^2 + 64c_n^2, 8c_n\big)$-sub-gamma.

Having established the sub-gamma property of KT-SPLIT halving, we can directly apply Theorem 1 to show that COMPRESS(KT-SPLIT, $\mathfrak{g}$) is also Γ_+^f sub-gamma with parameters

$$\sigma_{\rm CP}^2 \leq 237\cdot\tfrac{n}{4^{\mathfrak{g}}}\|f\|_{\mathbf{k}}^2\|\mathbf{k}\|_\infty(\log n + 3\mathfrak{g} + 3)(\log n - \mathfrak{g}), \quad \text{and} \quad c_{\rm CP} \leq 22\sqrt{\|\mathbf{k}\|_\infty}\|f\|_{\mathbf{k}},$$

which in turn implies that for $\mathfrak{g} = 0$, we have

$$\left|\tfrac{1}{n}\sum_{x\in\mathcal{S}_{\rm in}} f(x) - \tfrac{1}{\sqrt{n}}\sum_{x\in\mathcal{S}_{\rm CP}} f(x)\right| \leq \tfrac{22\|f\|_{\mathbf{k}}\sqrt{\|\mathbf{k}\|_\infty}}{\sqrt{n}}\left[\sqrt{(\log^2 n + 3\log n)\log\big(\tfrac{1}{\delta}\big)} + \log\big(\tfrac{1}{\delta}\big)\right]$$

with probability $1-\delta$, which is at most $\mathcal{O}(\sqrt{\log n})$ worse than the KT-SPLIT guarantee itself.

2.2 MMD GUARANTEES FOR COMPRESS

Next, we turn attention to establishing the worst-case function error, namely the MMD guarantees for COMPRESS. For brevity, we give simplified upper bounds on the error parameters. For the exact expressions and the proof, see App. D. Here $\mathcal{S}_{\rm CP}$ denotes the output of COMPRESS.

Theorem 2 (MMD Guarantees for COMPRESS) *Let* HALVE *be a halving algorithm such that for any size n input it satisfies*

$$\mathbb{P}\big[\mathrm{MMD}(\mathcal{S}_{\rm in}, \mathcal{S}_{\rm HALVE}) \geq a_n + \sqrt{2v_n t} + c_n t\big] \leq e^{-t} \textit{ for all } t \geq 0, \quad \textit{and} \quad \mathbb{E}\psi_{\rm HALVE} = 0.$$

Then, COMPRESS(HALVE, $\mathfrak{g}$) *with size n input satisfies*

$$\Pr\Big[\mathrm{MMD}(\mathcal{S}_{\rm in}, \mathcal{S}_{\rm CP}) \geq \sigma_{\rm M,CP}\cdot\sqrt{2t} + 2tR_{\rm M,CP}\Big] \leq 2(n+1)e^{-t} \quad \textit{for all} \quad t \geq 0, \tag{5}$$

where $R_{\rm M,CP} \triangleq \sqrt{8v_q + 2a_q^2} + 4c_q$, *and* $\sigma_{\rm M,CP} \triangleq R_{\rm M,CP}\sqrt{\log_4 n - \mathfrak{g}}$ *with* $q \triangleq \sqrt{n}2^{\mathfrak{g}+1}$.

Remark 4 (Symmetrization) We can convert any halving algorithm into one that satisfies the *mean zero* condition $\mathbb{E}\psi_{\rm HALVE} = 0$ without impacting integration error, by *symmetrization*, i.e., by returning either the outputted half or its complement with equal probability.

Example 2 (Kernel thinning meets COMPRESS.) For MMD guarantees, we analyze COMPRESS with kernel thinning of Dwivedi & Mackey (2021b, Alg 1.) as the halving algorithm with $\mathfrak{g} = 0$, so that the overall algorithm is root-thinning, We call this instantiation *KT-Compress*, and compare and contrast it to KT for root thinning. [1]

First, we note that KT has a quadratic runtime of order n^2, so that COMPRESS once again provides a near linear $\mathcal{O}(\frac{n}{\log n})$ speed-up by Rem. 2. Next, we claim that the MMD error for KT COMPRESS is at most $\mathcal{O}(\log n)$ worse compared to KT, a mild degradation compared to the linear speed up. To prove this, first we note that for an output $\mathcal{S}_{\rm KT}$ of size $\frac{n}{2^m}$ from KT, Dwivedi & Mackey (2021a, Thm. 2) guarantees that

$$\mathrm{MMD}(\mathcal{S}_{\rm in}, \mathcal{S}_{\rm KT}) \precsim_{\mathbf{k}} \tfrac{2^m}{n} + \tfrac{2^m}{n}\sqrt{\log n\cdot\mathcal{M}_{\mathbf{k}}(\mathcal{B}_2(R), \tfrac{2^m}{n})}\sqrt{\log(1/\delta)} + \tfrac{2^m}{n}\sqrt{\log n}\log(1/\delta), \tag{6}$$

with probability at least $1-\delta$, where $\mathcal{M}_{\mathbf{k}}(\mathcal{A}, \varepsilon)$ denotes the ε-L^∞ cover of the unit ball of RKHS with domain restricted to the set $\mathcal{A}$ (Dwivedi & Mackey (2021a, Def. 1)), and $R \triangleq \max_{x\in\mathcal{S}_{\rm in}}\|x\|_2$. Inverting this bound, we conclude that KT as a halving algorithm (i.e., $m = 1$) satisfies the assumption of Theorem 2 with

$$a_n \precsim \tfrac{1}{n}, \quad v_n \precsim_{\mathbf{k}} \tfrac{\log n}{n^2}\cdot\mathcal{M}_{\mathbf{k}}(\mathcal{B}_2(R), \tfrac{2}{n}), \quad \text{and} \quad c_n \precsim_{\mathbf{k}} \tfrac{\sqrt{\log n}}{n}.$$

[1] Throughout this work, we use KT to refer to the KT variant which uses target kernel $\mathbf{k}$ in both KT-SPLIT and KT-SWAP steps, a variant referred to as target KT by Dwivedi & Mackey (2021a).

Now applying Theorem 2, we find that KT compress admits the tail bound (5) with

$$R_{\mathrm{M,KT\text{–}CP}} \precsim_{\mathbf{k}} \sqrt{\log n \cdot \mathcal{M}_{\mathbf{k}}(\mathcal{B}_2(R), \tfrac{1}{\sqrt{n}})} \quad \text{and} \quad \sigma_{\mathrm{M,KT\text{–}CP}} \precsim_{\mathbf{k}} \log n \cdot \sqrt{\mathcal{M}_{\mathbf{k}}(\mathcal{B}_2(R), \tfrac{1}{\sqrt{n}})}.$$

Choosing $t^\star = \log(\frac{2(n+1)}{\delta})$, we conclude that the $\sqrt{n}$ sized KT-compress output $\mathcal{S}_{\mathrm{KT\text{–}CP}}$ satisfies

$$\mathrm{MMD}(\mathcal{S}_{\mathrm{in}}, \mathcal{S}_{\mathrm{KT\text{–}CP}}) \precsim_{\mathbf{k}} \sqrt{\log n}\Big(\sqrt{\log n \cdot \log(\tfrac{n}{\delta})} + \log(\tfrac{n}{\delta})\Big) \tfrac{1}{\sqrt{n}} \sqrt{\mathcal{M}_{\mathbf{k}}(\mathcal{B}_2(R), \tfrac{1}{\sqrt{n}})}, \tag{7}$$

with probability at least $1 - \delta$. On the other hand, with high probability the output from KT (when root thinning) itself satisfies $\frac{1}{\sqrt{n}}\sqrt{\log n \cdot \mathcal{M}_{\mathbf{k}}(\mathcal{B}_2(R), \frac{1}{\sqrt{n}})}\sqrt{\log(1/\delta)}$ where we note that this bound follows from (6) by substituting $m = \frac{1}{2}\log_2 n$. Comparing (7) to the above bound, we conclude that KT-COMPRESS provides at most $\mathcal{O}(\log n)$ worse MMD error than KT itself for general $\mathbf{k}$, and $\mathcal{S}_{\mathrm{in}}$.

3 COMPRESS ++

In this section, we introduce another simpleyet effective meta procedure—COMPRESS++—that uses a generic thinning algorithm along with COMPRESS in two stages. Overall, it provides a significantly improved error at the cost of a mild logarithmic overhead in the running time when compared to COMPRESS. Algorithmically, COMPRESS++ takes as input an α-thinning algorithm (Def. 2) THIN(α) for $\alpha \in \mathbb{N}$.[2] In the first stage, it runs COMPRESS with THIN(2) as the halving procedure and oversampling factor $\mathfrak{g}$ set to a non-zero quantity using THIN(2) as the halving algorithm to reduce the size from n to $\sqrt{n}2^{\mathfrak{g}}$. Next, we run the THIN($2^{\mathfrak{g}}$) on the output of the first stage to thin the sequence down to size $\sqrt{n}$. We formally present this algorithm in Alg. 2.

Algorithm 2: COMPRESS++

Input: Thinning algorithm THIN, Oversampling parameter $\mathfrak{g}$, Input point sequence $\mathcal{S}_{\mathrm{in}}$
$\mathcal{S}_{\text{COMPRESS}} \leftarrow$ COMPRESS(THIN(2), $\mathfrak{g}$, $\mathcal{S}_{\mathrm{in}}$) // Coreset of size $2^{\mathfrak{g}}\sqrt{n}$
$\mathcal{S}_{\text{COMPRESS++}} \leftarrow$ THIN($2^{\mathfrak{g}}$, $\mathcal{S}_{\mathrm{CP}}$) // Coreset of size $\sqrt{n}$
return $\mathcal{S}_{\mathrm{CP++}}$

3.1 SUB-GAMMA PROPERTY AND RUNTIME GUARANTEES FOR COMPRESS++

First, we note the quality of the solution given by COMPRESS++ by expressing the Γ_+^f parameters in terms of those of THIN. The proof can be found in App. I.

Theorem 3 (Sub-gamma property and runtime of COMPRESS++) *Let* THIN(α) *denote a* Γ_+^f-α*-thinning algorithm with parameters* $\sigma^2_{\mathrm{THIN}}(n,\alpha)$ *and* $c_{\mathrm{THIN}}(n,\alpha)$*, and running time* $r_{\mathrm{THIN}}(n,\alpha)$ *for inputs of size* n*. Then* COMPRESS++(THIN, $\mathfrak{g}$) *is root-thinning and* Γ_+^f *with parameters*

$$\begin{aligned}\sigma^2_{\mathrm{CP++}}(n,\mathfrak{g}) &= \tfrac{n}{4^{\mathfrak{g}+1}}\textstyle\sum_{i=0}^{\log_4 n - g} \sigma^2_{\mathrm{THIN}}\big(2^{\mathfrak{g}+1-i}\sqrt{n}, 2\big) + \tfrac{n}{4^{\mathfrak{g}}}\sigma^2_{\mathrm{THIN}}\big(2^{\mathfrak{g}}\sqrt{n}, 2^{\mathfrak{g}}\big)\\ &\le \tfrac{n}{4^{\mathfrak{g}+1}}(\log_4 n - \mathfrak{g})\sigma^2_{\mathrm{THIN}}\big(2^{\mathfrak{g}+1}\sqrt{n}, 2\big) + \tfrac{n}{4^{\mathfrak{g}}}\sigma^2_{\mathrm{THIN}}\big(2^{\mathfrak{g}}\sqrt{n}, 2^{\mathfrak{g}}\big)\\ c_{\mathrm{CP++}}(n,\mathfrak{g}) &= \max\Bigg(\tfrac{\sqrt{n}}{2^{\mathfrak{g}}}c_{\mathrm{THIN}}\big(2^{\mathfrak{g}}\sqrt{n}, 2^{\mathfrak{g}}\big), \max_{0\le i\le \log_4 n - g - 1}\Big(\tfrac{\sqrt{n}}{2^{\mathfrak{g}+i+1}}c_{\mathrm{THIN}}\big(\sqrt{n}2^{\mathfrak{g}+1-i}, 2\big)\Big)\Bigg),\end{aligned}$$

with its runtime bounded as

$$r_{\mathrm{CP++}}(n) = \textstyle\sum_{i=0}^{\log_4 n - \mathfrak{g} - 1} 4^i r_{\mathrm{THIN}}\big(2^{\mathfrak{g}+1-i}\sqrt{n}, 2\big) + r_{\mathrm{THIN}}\big(2^{\mathfrak{g}}\sqrt{n}, 2^{\mathfrak{g}}\big).$$

With some algebra with expressions above, one can see the advantage of COMPRESS++ over COMPRESS. Setting $g = \log_4 \log_4 n$, we can see that

$$\sigma^2_{\mathrm{CP++}}(n,\mathfrak{g}) \le n\sigma^2_{\mathrm{THIN}}\big(2\sqrt{n\log n}, 2\big) + \tfrac{n}{\log n}\sigma^2_{\mathrm{THIN}}\Big(\sqrt{n\log n}, \sqrt{\log_4 n}\Big)$$

[2] One can construct an α-thinning algorithm given a halving algorithm by recursively calling the halving algorithm on the output of the previous stage. See App. H for further discussion.

Normalizing by the size of the input, we get that the integration error of COMPRESS++ is comparable to error of just running THIN on sets of size $\sqrt{n \log n}$ to get an output of size $\sqrt{n}$ and the error of halving a data set of size $2\sqrt{n \log n}$. Thus, the integration error is comparable to that of methods such as **RecHalve** (discussed in App. H). And, we note that COMPRESS++ provides a logarithmic factor improvement in error over COMPRESS.

Example 3 (KT-SPLIT meets COMPRESS++) Example 1 showed that KT-SPLIT algorithm is $\Gamma_+^f\Big(32\sigma_{n,\alpha}^2 + 64c_{n,\alpha}^2, 8c_{n,\alpha}\Big)$ when the input size is n and the output size is n/α with $c_{n,\alpha} = \alpha \cdot \sqrt{\frac{32\log 4}{3}\|\mathbf{k}\|_\infty}\|f\|_{\mathbf{k}}$ and $\sigma_{n,\alpha}^2 = \alpha^2 \frac{4\log 4}{3}\log(2n) \cdot \|f\|_{\mathbf{k}}^2\|\mathbf{k}\|_\infty$. Thus, if we run COMPRESS++(KT-SPLIT, $\mathfrak{g}$), we get $\sigma_{\text{CP++}}^2 \leq \frac{4096\log 4}{3}\|f\|_{\mathbf{k}}^2\|\mathbf{k}\|_\infty\left[n4^{-\mathfrak{g}-1}(\log_4 n - g)\log(2^{g+2}\sqrt{n}) + n\log(2^{\mathfrak{g}}\sqrt{n})\right]$ and $c_{\text{CP++}} \leq \sqrt{\frac{1024\log 4}{3}\|f\|_{\mathbf{k}}^2\|\mathbf{k}\|_\infty}\sqrt{n}$. Setting, $g = \log_4\log_4 n$, gives us $\sigma_{\text{CP++}}^2$ which can be seen to be factor of two from the bound on the sub-gamma parameters for KT-SPLIT with $\alpha = \sqrt{n}$.

Optimal $\mathfrak{g}$: In this setting, we derive more precise bounds in App. J for optimal $\mathfrak{g}$ (to minimize the error) and note there that large sample values of $n = 10^7$, a default choice of $\mathfrak{g}_{opt} = 4$ suffices (which we also use in all our experiments). We highlight that with such a choice of g, the run-time is order $\mathcal{O}n\log^2 n$.

Next, we give MMD guarantees for COMPRESS++ in terms of the MMD guarantees of the thinning algorithm THIN. The proof can be found in App. K.

Theorem 4 (COMPRESS++ MMD Guarantees) *Let* THIN(α) *is an α-thinning algorithm such that on input $\mathcal{S}_{\text{in}}$ of size n outputs $\mathcal{S}_{\text{THIN}(\alpha)}$ such that*

$$\Pr\left[\text{MMD}\Big(\mathcal{S}_{\text{in}}, \mathcal{S}_{\text{THIN}(\alpha)}\Big) \geq a_{n,\alpha} + \sqrt{2v_{n,\alpha}t} + c_{n,\alpha}t\right] \leq e^{-t} \quad \textit{and} \quad \mathbb{E}\psi_{\text{THIN}(2)} = 0.$$

Then, the $\sqrt{n}$ sized output $\mathcal{S}_{\text{CP++}}$ of COMPRESS++(THIN, $\mathfrak{g}$) *satisfies*

$$\Pr\left[\text{MMD}(\mathcal{S}_{\text{in}}, \mathcal{S}_{\text{CP++}}) \geq a_s + \sqrt{2(\sigma_{\text{M,CP}}^2 + v_s)t} + (c_s + 2R_{\text{M,CP}})t\right] \leq 2(n+2)e^{-t},$$

where $s \triangleq (2^{\mathfrak{g}}\sqrt{n}, 2^{\mathfrak{g}})$, and $(R_{\text{M,CP}}, \sigma_{\text{M,CP}})$ were defined in Theorem 2.

Example 4 (Kernel thinning meets COMPRESS++.) We now illustrate the advantages of COMPRESS++, which we call *KT-Compress++*, defined as COMPRESS++ with kernel thinning Dwivedi & Mackey (2021b, Alg 1.) as the thinning algorithm. Let $\mathcal{S}_{\text{KT–CP}}$ and $\mathcal{S}_{\text{KT–CP++}}$ denote the output of the two stages of COMPRESS++.

Recall from Example 2 that

$$\text{MMD}(\mathcal{S}_{\text{in}}, \mathcal{S}_{\text{KT–CP}}) \precsim_{\mathbf{k}} \sqrt{\log n}\Big(\sqrt{\log n \cdot \log(\tfrac{n}{\delta})} + \log(\tfrac{n}{\delta})\Big)\tfrac{1}{2^{\mathfrak{g}}\sqrt{n}}\sqrt{\mathcal{M}_{\mathbf{k}}(\mathcal{B}_2(R), \tfrac{1}{2^{\mathfrak{g}}\sqrt{n}})},$$

with probability at least $1-\delta$. Also, recall that for thinning a set of size $2^g\sqrt{n}$ to $\sqrt{n}$, kernel thinning has the guarantee

$$\text{MMD}(\mathcal{S}_{\text{in}}, \mathcal{S}_{\text{KT}}) \precsim_{\mathbf{k}} \tfrac{1}{\sqrt{n}} + \tfrac{1}{\sqrt{n}}\sqrt{\log(2^g\sqrt{n}) \cdot \mathcal{M}_{\mathbf{k}}(\mathcal{B}_2(R), \tfrac{1}{\sqrt{n}})}\sqrt{\log(1/\delta)} + \tfrac{1}{\sqrt{n}}\sqrt{\log 2^g\sqrt{n}}\log(1/\delta),$$

with probability at least $1-\delta$, where $\mathcal{M}_{\mathbf{k}}(\mathcal{A}, \varepsilon)$ and R carry the same meaning as in Example 2. Thus, with probability at least $1-2\delta$, we have

$$\text{MMD}(\mathcal{S}_{\text{in}}, \mathcal{S}_{\text{KT–CP++}}) \leq \text{MMD}(\mathcal{S}_{\text{in}}, \mathcal{S}_{\text{KT–CP}}) + \text{MMD}(\mathcal{S}_{\text{KT–CP++}}, \mathcal{S}_{\text{KT–CP}})$$

Setting $2^g = \log n$, we get a bound of

$$\text{MMD}(\mathcal{S}_{\text{in}}, \mathcal{S}_{\text{KT–CP++}}) \precsim_{\mathbf{k}} 2\Big(\sqrt{\log(\tfrac{n}{\delta})} + \log(\tfrac{n}{\delta})\Big)\tfrac{1}{\sqrt{n}}\sqrt{\mathcal{M}_{\mathbf{k}}(\mathcal{B}_2(R), \tfrac{1}{\sqrt{n}\log n})},$$

On the other hand, with high probability the direct KT output of size $\sqrt{n}$ satisfies $\frac{1}{\sqrt{n}}\sqrt{\log n \cdot \mathcal{M}_{\mathbf{k}}(\mathcal{B}_2(R), \frac{1}{\sqrt{n}})}\sqrt{\log(1/\delta)}$ where we note that this bound follows from (6) by substituting m such that $n/2^m = 2^{\mathfrak{g}}\sqrt{n}$ (the output size). Comparing the bounds, we conclude that KT-COMPRESS provides a constant worse MMD error than KT itself for $\mathbf{k}$, and $\mathcal{S}_{\text{in}}$, when $\mathcal{M}_{\mathbf{k}}(\mathcal{B}_2(R), \frac{1}{\sqrt{n}}) = \Theta(\mathcal{M}_{\mathbf{k}}(\mathcal{B}_2(R), \frac{1}{\sqrt{n \log n}}))$. This condition is satisfied when $\mathcal{M}_{\mathbf{k}}(\mathcal{B}_2(R), \varepsilon) \precsim_d R^d(\log(1/\varepsilon))^{\omega}$ for suitable ω; and Dwivedi & Mackey (2021a, Prop. 3) shows that this bound on covering number is valid for analytic kernels like Gaussian and inverse multiquadrics.

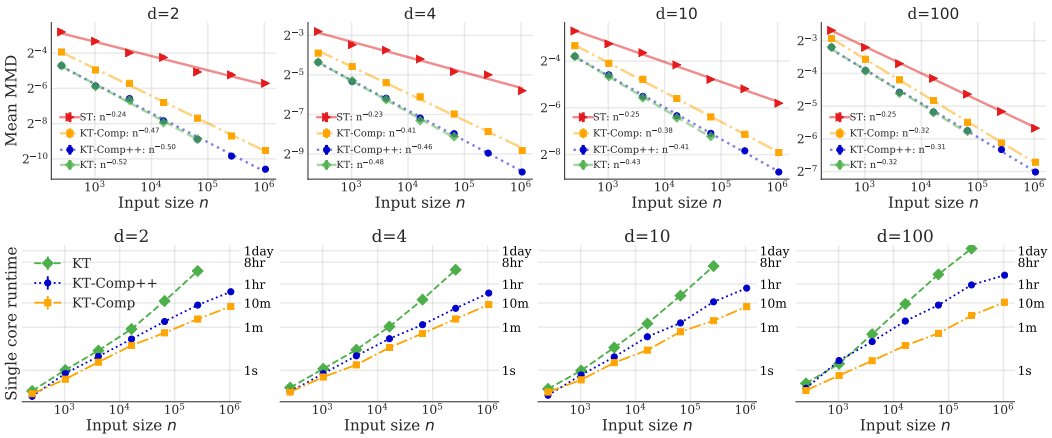

Figure 1: Gaussian Target Experiments with KT, and its COMPRESS and COMPRESS++ variants, with MMD error plotted in first row along with with the runtime plotted in second row.

4 EXPERIMENTS

We now provide experiments (with additional experiments and suppmentary details in App. L) that demonstrate the speed ups with minimal loss of error in several experiments. We compare KT and kernel herding with their COMPRESS and COMPRESS++ versions, and in all tasks we do root thinning, namely take n input points and compress it down to $\sqrt{n}$ points. For COMPRESS++ we set $\mathfrak{g} = 4$ in all experiments. We evaulate the procedures both in terms of MMD error (averaged across 10 runs), and runtime (averaged across 3 runs); the error bars are too small to be visible in almost all settings. All timing experiments were run on a single core of an Intel Xeon CPU with 32GB RAM.

Target distributions and experiment settings We consider three sets of target distributions: (a) Gaussian target $\mathbb{P} = \mathcal{N}(0, \mathbf{I}_d)$ for $d \in \{2, 4, 10, 100\}$, (b) M-Mixture of Gaussian target $\mathbb{P} = \frac{1}{M}\sum_{j=1}^{M}\mathcal{N}(\mu_j, \mathbf{I}_2)$ with $M \in \{4, 6, 8, 31\}$ with component means $\mu_j \in \mathbb{R}^2$ defined in App. L, and (c) MCMC target for 12 different Bayesian posterior inference tasks with data taken from Riabiz et al. (2020a) and additional details provided in App. L. All our experiments are run with Gaussian kernel $\mathbf{k}(x, y) = \exp(-\frac{1}{2\sigma^2}\|x - y\|_2^2)$ with σ^2 set to $2d$ for (a), and (b), and chosen using median heuristic over the entire MCMC chain Garreau et al. (2017) for (c). For error performance, we evaluate the different thinning algorithms with respect to the MMD metric $\mathrm{MMD}_{\mathbf{k}}$ for (a) and (b) since it can be computed in closed-form, and for (c) since the posterior is intractable, we measure the quantity we bound namely $\mathrm{MMD}_{\mathbf{k}}(\mathbb{P}_{\text{in}}, \mathbb{P}_{\text{out}})$—the error between the input and output empirical distributions. For (a) and (b), the input sequence is generated i.i.d. from the target $\mathbb{P}$, and for (c) we first standard thin the chain to get n input points which is then fed to the different thinning algorithms (see App. L for details). For all error plots, we also provide the OLS fit on the log-log scale for the error and the input size, and its slope is shown as the emprical decay rate, e.g., a slope of -0.48 for the best fit is labeled as $n^{-0.48}$ for the corresponding algorithm.

Kernel thinning experiments We evaluate standard thinning (ST), kernel thinning (KT), KT-COMPRESS (denoted as KT-Comp), KT-COMPRESS++ (denoted as KT-Comp++) for all the settings (a), (b) and (c), and we present the results in Figs. 1, 2, and 6 respectively. We note that standard thinning provides close to the Monte Carlo rate of $n^{-\frac{1}{4}}$ with $n^{\frac{1}{2}}$ points in almost all the experiments,

while KT and its COMPRESS++ variants provide the best performance often achieving the $n^{-\frac{1}{2}}$ rate as guaranteed by our theory in moderate dimensions. The runtimes for the procedures are compared in Fig. 1 where we note that COMPRESS++ provides a noticeable 10-25x speed up over the KT runtime (and compress even more), consistent with our theoretical guarantees. In fact, the faster runtime of COMPRESS and COMPRESS++ allowed us to run experiments with them to a significantly larger input size for which we KT took many days to finish a single run (and thereby was not run).

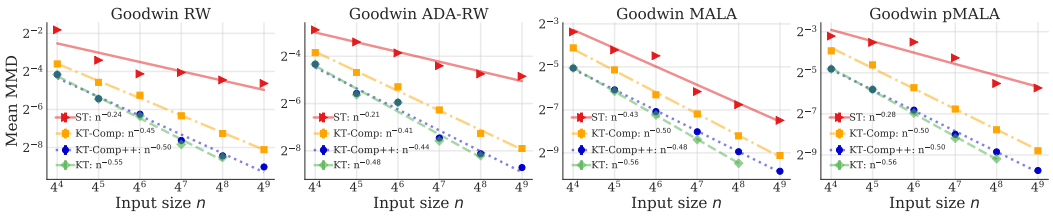

Figure 2: 4 MCMC target experiments with KT, and its COMPRESS and COMPRESS++ variants

Kernel herding experiments As another demonstration of the practical advantages of COMPRESS and COMPRESS++, we apply these procedures with kernel herding Chen et al. (2010) for Gaussian and MoG targets and report the results in Figs. 3 and 7. The results are labeled as kernel herding (Herd), COMPRESS herding (Herd-Comp), and COMPRESS++ with herding (Herd-Comp++). In all cases, we observe that Herd-Comp++ provides a competitive performance to herding, sometimes improving upon it. Fig. 3 shows COMPRESS++ with herding is $10 - 60$x faster than herding for large samples sizes thereby providing further evidence about the compatibility of our meta procedures with generic thinning procedures, and their practical impact on saving computation time with minimal loss in accuracy.

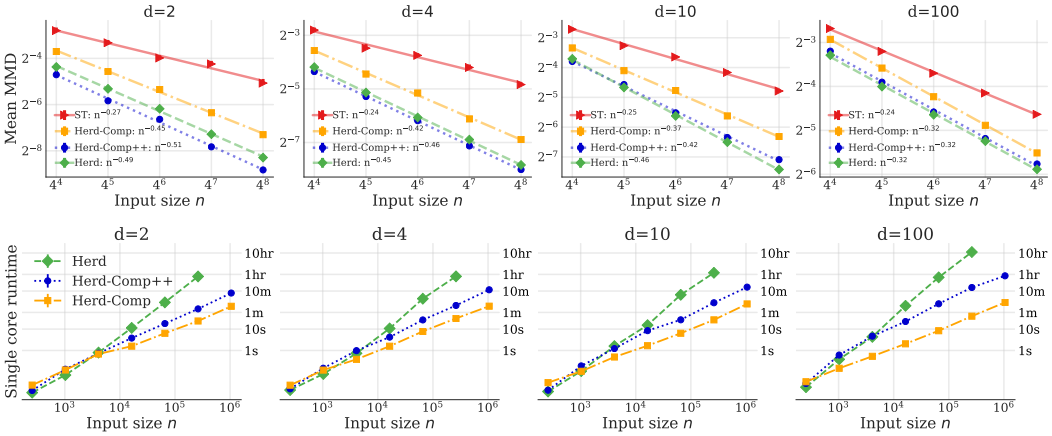

Figure 3: Gaussian target experiments with kernel herding and its COMPRESS and COMPRESS++ variants, with error plots provided in the top row, and the runtime provided in the bottom row.

5 CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we introduce an easy to implement and versatile meta-algorithms to speed up distribution compression tasks at almost no cost to the error guarantees. We have demonstrated, both theoretically and empirically that the performance on benchmark task are comparable to that of previous work. Furthermore, we have shown that these meta-algorithms even work well in practice with algorithms that don't have theoretical guarantees but work well in practice e.g. herding. Analyzing the behavior of these meta-algorithms in other compression setting is an interesting direction for future work.

REPRODUCIBILITY STATEMENT

Our Python code reconstructing all experiments can be found in a supplementary zip file.

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

Anaël Beaugnon, Pierre Chifflier, and Francis Bach. Ilab: An interactive labelling strategy for intrusion detection. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 120–140. Springer, 2017.

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.

W. Y. Chen, L. Mackey, J. Gorham, F-X. Briol, and C. J. Oates. Stein points. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Wilson Ye Chen, Alessandro Barp, François-Xavier Briol, Jackson Gorham, Mark Girolami, Lester Mackey, and Chris Oates. Stein point Markov chain Monte Carlo. In *International Conference on Machine Learning*, pp. 1011–1021. PMLR, 2019.

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, pp. 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.

Damien Garreau, Wittawat Jitkrittum, and Motonobu Kanagawa. Large sample analysis of the median heuristic. *arXiv preprint arXiv:1707.07269*, 2017.

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.

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

Simon Mak and V Roshan Joseph. Support points. *The Annals of Statistics*, 46(6A):2562–2592, 2018.

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*, pp. 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

1 Introduction **1**

2 Compress **3**

2.1 Sub-gamma property and runtime guarantees for COMPRESS 3

2.2 MMD guarantees for COMPRESS 5

3 Compress ++ **6**

3.1 Sub-gamma property and runtime guarantees for COMPRESS++ 6

4 Experiments **8**

5 Conclusions and Future Directions **9**

A Properties of Sub-Gamma Random Variables **12**

B Proof of Theorem 1: Sub-gamma property and runtime of COMPRESS **13**

C COMPRESS Calculations **14**

D Proof of Theorem 2: MMD Guarantees for COMPRESS **14**

E Proof of Lem. 2: Proof of Exponential Matrix Freedman Inequality **16**

F Proof of Lem. 3: Tail Bounds to Moments **16**

G Streaming Version of COMPRESS **17**

H Guarantees for RecHalve **18**

I Proof of Theorem 3: Sub-gamma property and runtime of COMPRESS++ **19**

J COMPRESS++ Calculations **19**

K Proof of Theorem 4: COMPRESS++ MMD Guarantees **20**

L Additional experiments. and supplementary information **21**

A PROPERTIES OF SUB-GAMMA RANDOM VARIABLES

Lemma 1 (Summation and scaling preserve Γ_+^f) *Let $f \in \mathcal{H}$ and $G_1 \in \Gamma_+^f(\sigma_1^2, c_1)$. Then, for all $\beta \geq 0$, we have $\beta \cdot G_1 \in \Gamma_+^f(\beta^2\sigma^2, \beta c)$. Furthermore, if $G_2 \in \Gamma_+^f(\sigma_2^2, c_2)$ conditioned on G_1, then $G_1 + G_2 \in \Gamma_+^f(\sigma_1^2 + \sigma_2^2, \max(c_1, c_2))$.*

A.1 PROOF OF LEM. 1: SUMMATION AND SCALING PRESERVE Γ_+^f

First, let us consider scaling. For any $k > 0$, since $\langle f, \cdot \rangle_{\mathcal{H}}$ is linear, we have

$$\langle f, k \cdot G_1 \rangle_{\mathcal{H}} = k \cdot \langle f, G_1 \rangle_{\mathcal{H}}.$$

Thus, for any $0 < \lambda \leq \frac{1}{kc}$,

$$\mathbb{E}\Big[\exp\big(\lambda \cdot \langle f, k \cdot G_1\rangle_{\mathcal{H}}\big)\Big] \leq \frac{k^2\sigma_1^2\lambda^2}{2(1-kc_1\lambda)}$$

as required.

Furthermore, note that

$$\langle f, G_1 + G_2\rangle_{\mathcal{H}} = \langle f, G_1\rangle_{\mathcal{H}} + \langle f, G_2\rangle_{\mathcal{H}}.$$

Let $0 < \lambda \leq \min\Big(\frac{1}{c_1}, \frac{1}{c_2}\Big)$. Then, since G_2 is conditionally sub-gamma given G_1, we have

$$\begin{aligned}
\mathbb{E}\Big[\exp\big(\lambda \cdot \langle f, G_1 + G_2\rangle_{\mathcal{H}}\big)\Big] &= \mathbb{E}\Big[\exp\big(\lambda \cdot \langle f, G_1\rangle_{\mathcal{H}} + \lambda \cdot \langle f, G_2\rangle_{\mathcal{H}}\big)\Big] \\
&= \mathbb{E}\Big[\mathbb{E}\Big[\exp\big(\lambda \cdot \langle f, G_1\rangle_{\mathcal{H}} + \lambda \cdot \langle f, G_2\rangle_{\mathcal{H}}\big)|G_1\Big]\Big] \\
&= \mathbb{E}\Big[\mathbb{E}\Big[\exp\big(\lambda \cdot \langle f, G_1\rangle_{\mathcal{H}}\big)\exp\big(\lambda \cdot \langle f, G_2\rangle_{\mathcal{H}}\big)|G_1\Big]\Big] \\
&= \mathbb{E}\Big[\exp\big(\lambda \cdot \langle f, G_1\rangle_{\mathcal{H}}\big) \cdot \mathbb{E}\Big[\exp\big(\lambda \cdot \langle f, G_2\rangle_{\mathcal{H}}\big)|G_1\Big]\Big] \\
&\leq \exp\Big(\frac{\lambda^2\sigma_2^2}{2(1-c_2\lambda)}\Big) \cdot \mathbb{E}\Big[\exp\big(\lambda \cdot \langle f, G_1\rangle_{\mathcal{H}}\big)\Big] \\
&= \exp\Big(\frac{\lambda^2\sigma_1^2}{2(1-c_1\lambda)}\Big) \cdot \exp\Big(\frac{\lambda^2\sigma_2^2}{2(1-c_2\lambda)}\Big) \\
&\leq \exp\Bigg(\frac{\lambda^2\big(\sigma_1^2+\sigma_2^2\big)}{2\big(1-\max\{c_1,c_2\}\lambda\big)}\Bigg)
\end{aligned}$$

as required.

B Proof of Theorem 1: Sub-gamma property and runtime of Compress

Note that

$$\psi_{\textsc{Compress}}(\mathcal{S}_{\text{in}}; 2^{\mathfrak{g}}) = \textstyle\sum_{j=1}^{4} \psi_{\textsc{Compress}}(\mathcal{S}_i; \mathfrak{g}) + \sqrt{n}2^{-\mathfrak{g}-1}\psi_{\textsc{Halve}}(\widetilde{\mathcal{S}})$$

where $\mathcal{S}_i$ and $\widetilde{\mathcal{S}}$ are defined as in Alg. 1. Let $\big(\sigma^2_{\textsc{Compress}}(n, \mathfrak{g}), c_{\textsc{Compress}}(n, \mathfrak{g})\big)$ and $\big(\sigma^2_{\textsc{Halve}}(n), c_{\textsc{Halve}}(n)\big)$ respectively denote the $\Gamma_+^{\mathcal{F}}$ parameters of $\psi_{\textsc{Compress}}(\cdot; \mathfrak{g})$, and $\psi_{\textsc{Halve}}(\cdot)$ on input of size n. Since $\psi_{\textsc{Compress}}(\mathcal{S}_i; \mathfrak{g})$ and $\psi_{\textsc{Halve}}(\widetilde{\mathcal{S}})$ are independent conditioned on the input, we have, using Lem. 1,

$$\sigma^2_{\textsc{Compress}}(n, 2^{\mathfrak{g}}) = 4\sigma^2_{\textsc{Compress}}\left(\frac{n}{4}, 2^{\mathfrak{g}}\right) + n4^{-\mathfrak{g}-1}\sigma^2_{\textsc{Halve}}\left(\sqrt{n}2^{\mathfrak{g}+1}\right)$$

and

$$c_{\textsc{Compress}}(n, 2^{\mathfrak{g}}) = \max\left(\sqrt{n}2^{-\mathfrak{g}-1}c_{\textsc{Halve}}\left(\sqrt{n}2^{\mathfrak{g}+1}\right), c_{\textsc{Compress}}\left(\frac{n}{4}, 2^{\mathfrak{g}}\right)\right)$$

Also, note that we can take $\sigma^2_{\textsc{Compress}}(4^{\mathfrak{g}}, 2^{\mathfrak{g}}) = c_{\textsc{Compress}}(4^{g}, 2^{\mathfrak{g}}) = 0$ as Compress on input a point sequence of size 4^g returns the same point sequence. Thus,

$$\sigma^2_{\textsc{Compress}}(n, 2^{\mathfrak{g}}) = n4^{-\mathfrak{g}-1}\sum_{i=0}^{\log_4 n-\mathfrak{g}-1}\sigma^2_{\textsc{Halve}}\left(2^{\mathfrak{g}+1}\sqrt{n4^{-i}}\right)$$

and

$$c_{\textsc{Compress}}(n, 2^{\mathfrak{g}}) = \max_{0\leq i\leq \log_4 n-g-1}\left(\sqrt{n}2^{-\mathfrak{g}-i-1}c_{\textsc{Halve}}\left(\sqrt{n}2^{\mathfrak{g}+1-i}\right)\right)$$

as required.

Next, we bound the running time of COMPRESS. Let $r_{\mathrm{CP}}(n)$ be the running time of the algorithm on input of size n. From the definition, COMPRESS makes four recursive calls to COMPRESS on inputs of size $n/4$. Then, HALVE is run on an input of size $2^{\mathfrak{g}+1}\sqrt{n}$. Thus, r satisfies the recursion

$$r_{\mathrm{CP}}(n) = 4r_{\mathrm{CP}}(n/4) + r_{\mathrm{HALVE}}(\sqrt{n}2^{\mathfrak{g}+1}).$$

Note that $r_{\mathrm{CP}}(4^{\mathfrak{g}}) = 1$. Unrolling the recursion, we get

$$r_{\mathrm{CP}}(n) = n4^{-\mathfrak{g}} + \sum_{i=0}^{\log_4 n-\mathfrak{g}-1} 4^i r_{\mathrm{HALVE}}(2^{\mathfrak{g}+1}\sqrt{n4^{-i}})$$

as required.

C COMPRESS CALCULATIONS

To get a sense for the guarantees given by the above algorithm, we will instantiate this for various settings for $\sigma^2_{\mathrm{HALVE}}$. In particular, natural choices for σ_{HALVE} are constant and logarithmic growth.

- $\sigma^2_{\mathrm{HALVE}}(n) = C\log_4 n$

$$\begin{aligned}\sigma^2_{\mathrm{CP}}(n,\mathfrak{g}) &= C\tfrac{n}{4^{\mathfrak{g}+1}} \cdot \textstyle\sum_{i=0}^{\log_4 n-\mathfrak{g}-1} \log_4\left(2^{\mathfrak{g}+1}\sqrt{n4^{-i}}\right)\\ &= C\tfrac{n}{4^{\mathfrak{g}+1}} \cdot \tfrac{1}{2}\textstyle\sum_{i=0}^{\log_4 n-\mathfrak{g}-1}(\log_4 n + \mathfrak{g} + 1 - i)\\ &= C\tfrac{n}{4^{\mathfrak{g}+1}} \cdot \tfrac{1}{2}\textstyle\sum_{j=2\mathfrak{g}+2}^{\log_4 n+\mathfrak{g}+1} j\\ &\overset{(i)}{=} C\tfrac{n}{4^{\mathfrak{g}+1}} \cdot \tfrac{1}{4}(\log_4 n + 3\mathfrak{g} + 3)(\log_4 n - \mathfrak{g}),\end{aligned}$$

- $\sigma^2_{\mathrm{HALVE}}(n) = C$

$$\sigma^2_{\mathrm{CP}}(n,\mathfrak{g}) = Cn4^{-\mathfrak{g}-1}(\log_4 n - \mathfrak{g}).$$

D PROOF OF THEOREM 2: MMD GUARANTEES FOR COMPRESS

By the definition of the COMPRESS algorithm, the COMPRESS discrepancy vector (2) satisfies

$$\psi_{\mathrm{CP}}(\mathcal{S},\mathfrak{g}) = \sqrt{n}2^{-\mathfrak{g}-1}\psi_{\mathrm{HALVE}}(\tilde{\mathcal{S}}, g) + \textstyle\sum_{i=1}^4 \psi_{\mathrm{CP}}(\mathcal{S}_i,\mathfrak{g}).$$

Unrolling the recursion, we find that HALVE is applied to 4^k coresets of size $\sqrt{n}2^{\mathfrak{g}+1}2^{-k}$. If we denote these coresets by $\tilde{\mathcal{S}}_{k,1},\cdots,\tilde{\mathcal{S}}_{k,4^k}$, we have

$$\psi_{\mathrm{CP}} = \sqrt{n}2^{-\mathfrak{g}-1}\textstyle\sum_{k=0}^{\log_4 n-g-1}\sum_{j=1}^{4^k} 2^{-k}\psi_{\mathrm{HALVE}}(\tilde{\mathcal{S}}_{k,j},\mathfrak{g}). \tag{8}$$

For each (k,j), let

$$\tilde{\psi}_{k,j} = \psi_{\mathrm{HALVE}}(\tilde{\mathcal{S}}_{k,j},\mathfrak{g})$$

represent the HALVE discrepancy vector. Then, for each l, the variables $(\tilde{\psi}_{l,j})_{j\geq 1}$ are conditionally independent and zero mean given $(\tilde{\psi}_{k,j})_{k>l,j\geq 1}$.

Moreover, if we number the elements of $\mathcal{S}_{\mathrm{in}}$ as $(x_1,\ldots,x_n)$,

$$\begin{aligned}\|\tilde{\psi}_{k,j}\|^2_{\mathbf{k}} = u^\top_{\tilde{\psi}_{k,j}} K u_{\tilde{\psi}_{k,j}} &= \left\|K^{1/2}u_{\tilde{\psi}_{k,j}}\right\|_2^2 \quad \text{for}\\ K &= (\mathbf{k}(x_i,x_j))_{1\leq i,j\geq n} \quad \text{and}\\ u_{\tilde{\psi}_{k,j}} &= \textstyle\sum_{i=1}^n e_i(1 - \tfrac{n}{n_{out}}\mathbb{I}[x_i\in\mathcal{S}_{\mathrm{out}}]) \in \mathbb{R}^n.\end{aligned}$$

For each $u\in\mathbb{R}^n$, consider the symmetric dilation $M_u \in \mathbb{R}^{n+1\times n+1}$ given by

$$\begin{pmatrix} 0 & u^T\\ u & 0\end{pmatrix}.$$

We will abbreviate $M_{u_{\tilde{\psi}_{k,j}}}$ as $M_{k,j}$. We will bound the spectral norm of this matrix in order to control $\|\tilde{\psi}_{k,j}\|_{\mathbf{k}}$. In order to do this, we will use the following subexponential version of the Matrix Freedman inequality of (Tropp, 2011, Thm. 1.2), a proof of which can be found in App. E.

Lemma 2 (Exponential Matrix Freedman Inequality) *Consider a finite sequence of random matrices $X_k \in \mathbb{R}^{d\times d}$ on that are self-adjoint. Let*

$$\mathbb{E}\Big[X_k|\{X_i\}_{i<k}\Big] = 0$$
$$\mathbb{E}\Big[X_k^p|\{X_i\}_{i<k}\Big] \preceq \tfrac{p!}{2}R^{p-2}A_k^2.$$

Then, if

$$\sigma^2 = \big\|\textstyle\sum A_k^2\big\|$$

Then, for all $t > 0$

$$\Pr\Big[\lambda_{\max}\big(\textstyle\sum_k X_k\big) \geq t\Big] \leq d\exp\Big(\tfrac{-t^2}{2(\sigma^2+Rt)}\Big).$$

First note that $\|\psi_{\text{CP}}\|_{\mathbf{k}} = n\cdot \text{MMD}(\mathcal{S}_{\text{in}}, \mathcal{S}_{\text{CP}})$. So it suffice to bound $\|\psi_{\text{CP}}\|_{\mathbf{k}}$. We will do this using Lem. 2. Note that for all the summands in (8) , $\mathbb{E}M_{k,j} = 0$ and furthermore,

$$M_{k,j}^{2q} = \|\tilde{\psi}_{k,j}\|_{\mathbf{k}}^{2q-2}M_{k,j}^2 \preceq \|\tilde{\psi}_{k,j}\|_{\mathbf{k}}^{2q}I$$
$$M_{k,j}^{2q+1} = \|\tilde{\psi}_{k,j}\|_{\mathbf{k}}^{2q-1}M_{k,j} \preceq \|\tilde{\psi}_{k,j}\|_{\mathbf{k}}^{2q+1}I$$

Therefore, $\mathbb{E}M_{k,j}^q \preceq \mathbb{E}\|\tilde{\psi}_{k,j}\|_{\mathbf{k}}^q I$. By assumption, we have that HALVE satisfies

$$\Pr\big[\text{MMD}(\mathcal{S}_{\text{in}}, \mathcal{S}_{\text{HALVE}}) \geq a_n + \sqrt{2v_n t} + c_n t\big] \leq e^{-t}.$$

on input sequences of size n. This implies that $\Pr\Big[\|\tilde{\psi}_{ell,j}\|_{\mathbf{k}} \geq a'_\ell + \sqrt{2v'_\ell t} + c'_\ell t\Big] \leq e^{-t}$ for $a'_\ell = \ell a_\ell, v'_\ell = \ell^2 v_\ell, c'_\ell = \ell c_\ell$ for $\tilde{\psi}_{k,j}$ of size $\ell = 2^{-k+g+1}\sqrt{n}$. We will use the following result that translates tail bounds into moment inequalities. A proof can be found in App. E.

Lemma 3 (Tail Bounds to Moments) *Let Z be a non-negative random variable such that*

$$\Pr\Big[Z > a + \sqrt{2vt} + ct\Big] \leq e^{-t}.$$

Then,

$$\mathbb{E}Z^q \leq \Gamma\big(\tfrac{q}{2}+1\big)(8v+2a^2)^{q/2} + (q)!(4c)^q.$$

For $\tilde{\psi}_{k,j}$, this implies $\mathbb{E}\|\tilde{\psi}_{k,j}\|_{\mathbf{k}}^q \leq \Gamma(q/2+1)(8v'_\ell + 2a_\ell^{'2})^{q/2} + (q)!(4c'_\ell)^q$ for $\ell = 2^{\mathfrak{g}+1-k}\sqrt{n}$. Thus, $\mathbb{E}\|w_{\tilde{\psi}_{k,j}}^q M_{k,j}^q\|_{\mathbf{k}} \preceq \frac{q!}{2}R_\ell^{q-2}A_\ell^2$ for $R_\ell = w_{\tilde{\psi}_{k,j}}(\sqrt{8v'_\ell + 2a_\ell^{'2}} + 4c'_\ell)$ and $A_\ell = 2w_{\tilde{\psi}_{k,j}}(\sqrt{8v'_\ell + 2a_\ell^{'2}} + 4c'_\ell)I$ where $w_{\tilde{\psi}_{k,j}}$ is the coefficient of $\tilde{\psi}_{k,j}$ in (8).

Thus, from Lem. 2, we have ψ_{CP} has exponential tails i.e.

$$\Pr\big[\|\psi_{\text{CP}}\|_{\mathbf{k}} \geq t\big] \leq 2(n+1)\exp\Big(\tfrac{-t^2}{2(\sigma^2+Rt)}\Big)$$

with

$$\begin{aligned}
\sigma^2 &= \textstyle\sum_{k=0}^{\log_4 n-g-1}\sum_{j=1}^{4^k} n2^{-2k-2g-2}\Bigg(\sqrt{8v'_{\sqrt{n}2^{\mathfrak{g}+1-k}} + 2a_{\sqrt{n}2^{\mathfrak{g}+1-k}}^{'2}} + 4c'_{\sqrt{n}2^{\mathfrak{g}+1-k}}\Bigg)^2 \\
&= \textstyle\sum_{k=0}^{\log_4 n-g-1} n2^{-2g-2}\Bigg(\sqrt{8v'_{\sqrt{n}2^{\mathfrak{g}+1-k}} + 2a_{\sqrt{n}2^{\mathfrak{g}+1-k}}^{'2}} + 4c'_{\sqrt{n}2^{\mathfrak{g}+1-k}}\Bigg)^2 \\
&\leq \Bigg(\sqrt{8v'_{\sqrt{n}2^{\mathfrak{g}+1}} + 2a_{\sqrt{n}2^{\mathfrak{g}+1}}^{'2}} + 4c'_{\sqrt{n}2^{\mathfrak{g}+1}}\Bigg)^2 \textstyle\sum_{k=0}^{\log_4 n-g-1} n2^{-2g-2} \\
&\leq \Bigg(\sqrt{8v'_{\sqrt{n}2^{\mathfrak{g}+1}} + 2a_{\sqrt{n}2^{\mathfrak{g}+1}}^{'2}} + 4c'_{\sqrt{n}2^{\mathfrak{g}+1}}\Bigg)^2 n2^{-2g-2}(\log_4 n - \mathfrak{g})
\end{aligned}$$

and

$$\begin{aligned}
R &= \max_{k\leq \log_4 n-\mathfrak{g}-1} \sqrt{n}2^{-\mathfrak{g}-1-k}\Bigg(\sqrt{8v'_{\sqrt{n}2^{\mathfrak{g}+1-k}} + 2a_{\sqrt{n}2^{\mathfrak{g}+1-k}}^{'2}} + 4c'_{\sqrt{n}2^{\mathfrak{g}+1-k}}\Bigg) \\
&\leq \sqrt{n}2^{-\mathfrak{g}-1}\Bigg(\sqrt{8v'_{\sqrt{n}2^{\mathfrak{g}+1}} + 2a_{\sqrt{n}2^{\mathfrak{g}+1}}^{'2}} + 4c'_{\sqrt{n}2^{\mathfrak{g}+1}}\Bigg).
\end{aligned}$$

We get the required bound by inverting expression in the exponential.

E Proof of Lem. 2: Proof of Exponential Matrix Freedman Inequality

Lemma 4 *Consider a finite sequence of random matrices $X_k \in \mathbb{R}^{d\times d}$ that are self-adjoint. Let*

$$\mathbb{E}\Big[X_k|\{X_i\}_{i<k}\Big] = 0$$
$$\mathbb{E}\Big[X_k^p|\{X_i\}_{i<k}\Big] \preceq \tfrac{p!}{2}R^{p-2}A_k^2.$$

Then, if

$$\sigma^2 = \big\|\sum A_k^2\big\|$$

Then, for all $t > 0$

$$\Pr\Big[\lambda_{\max}\big(\textstyle\sum_k X_k\big) \geq t\Big] \leq d\exp\Big(\tfrac{-t^2}{2(\sigma^2+Rt)}\Big).$$

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

Lemma 5 (Master Tail Bounds (Tropp, 2011)) *Let X_k be an adaptive sequence of matrix valued random variables in $\mathbb{R}^d$ (with respect to the filtration $\mathcal{F}_k$ and let V_k be a sequence that is measureable with respect to $\mathcal{F}_{k-1}$. Assume that*

$$\log\mathbb{E}\big[\exp(\theta X_k)|\mathcal{F}_{k-1}\big] \preceq g(\theta)V_k$$

Let $Y_k = \sum_{i\leq k} X_i$ and $W_k = \sum_{i\leq k} V_i$. Then,

$$\Pr\big[\exists k > 0 : \lambda_{\max}(Y_k) \geq t \text{ and } \lambda_{\max}(W_k) \leq w\big] \leq d\exp\big(-\theta t + g(\theta)w\big).$$

Lemma 6 (Matrix MGF Exponential (Tropp, 2012)) *Let X be a random self-adjoint matrix such that*

$$\mathbb{E}X = 0$$
$$\mathbb{E}X^q \preceq \tfrac{p!}{2}A^2 \text{ for } q = 2, 3, \ldots$$

Then,

$$\mathbb{E}\exp(\theta X) \preceq \exp\Big(\tfrac{\theta^2}{2(1-\theta)}\cdot A^2\Big) \text{ for } 0 \leq \theta \leq 1.$$

Assume that $R = 1$. We get the general result by scaling. We apply Lem. 6 conditional on $\{X_i\}_{i<k}$ along with the operator monotonicity of $\log$ to get

$$\log\mathbb{E}\Big[\exp(\theta X_k)|\{X_i\}_{i<k}\Big] \preceq \tfrac{\theta^2}{2(1-\theta)}\cdot A^2$$

By assumption, we have that $\lambda_{max}\big(\sum_i A_i^2\big) \leq \sigma^2$. Thus, applying Lem. 5, we get

$$\Pr\Big[\lambda_{\max}\big(\textstyle\sum_i X_i\big)\Big] \leq d\exp\Big(-\theta t + \tfrac{\theta^2\sigma^2}{2(1-\theta)}\Big)$$

We get the result by choosing $\theta = t/(t+\sigma^2)$.

F Proof of Lem. 3: Tail Bounds to Moments

We bound the moments of the random variable $X = Z - a$. First note that $X = X_+ - X_-$ where $X_\pm = \max(\pm X, 0)$. Also, $|X|^q = X_+^q + X_-^q$. Furthermore, $X_-^q \leq a^q$ by assumption.

$$\mathbb{E}\big[|X|^q\big] \leq a^q + \mathbb{E}\big[X_+^q\big].$$

In order to bound the moments of X_+, we use the tail bounds on X. Note that $\Pr[X_+ > t] = \Pr[X > t]$ for any $t > 0$.

$$\begin{aligned}
\mathbb{E}\left[X_+^q\right] &= q\int_0^\infty t^{q-1}\Pr[X > t]dt \\
&\le q\int_0^\infty (\sqrt{2vt} + ct)^{q-1}\left(\tfrac{\sqrt{2vt}+ct}{2t}\right)\tfrac{e^{-t}}{t}dt \\
&\le \tfrac{q}{2}\int_0^\infty (\sqrt{2vt} + ct)^{q}\tfrac{e^{-t}}{t}dt \\
&\le q2^{q-2}\int_0^\infty \Big((2tv)^{q/2} + (2ct)^q\Big)e^{-t}dt \\
&\le 2^{q-2}\Big(\Gamma(q/2+1)2^{q/2}v^{q/2} + \tfrac{(q)!}{2}(2c)^q\Big) \\
&\le \Gamma(q/2+1)(8v)^{q/2} + (q)!(4c)^q.
\end{aligned}$$

Thus, we have $\mathbb{E}Z^q \le 2^{q-1}(a^q + \mathbb{E}|X|^q)$ which gives us the desired result.

G Streaming Version of Compress

Compress can be efficiently implemented in a streaming fashion, by unrolling the recursion in the original Compress and looking at the algorithm from a bottom up point of view. In this setup, the algorithm maintains memory in layers, and takes in inputs in batches of size $4^{\mathfrak{g}+1}$. Whenever there is more than $2^i 4^{\mathfrak{g}+1}$ points in layer i, the algorithm uses Halve on the points in the layer and moves the points up to level $i+1$. The points in the highest non-empty layer form the current approximation to the points seen so far. See Alg. 3 for formal description.

Algorithm 3: Compress (Streaming)

Input: Halving algorithm Halve, oversampling factor $\mathfrak{g}$, input stream $x_1, x_2, \ldots$
$\mathcal{S}_0 \leftarrow \{\}$
for $t = 1, 2, \ldots,$ **do**
  $\mathcal{S}_0 \leftarrow \mathcal{S}_0 \cup (x_j)_{j=1+(t-1)\cdot 4^{\mathfrak{g}+1}}^{t\cdot 4^{\mathfrak{g}+1}}$ // Process input in batches of size $4^{\mathfrak{g}+1}$
  if $t == 4^j$ **then**
    $\mathcal{S}_{j+1} \leftarrow \{\}$
  end
  for $i = 0, \ldots, \log_4 t + 1$ **do**
    if $|\mathcal{S}_i| == 2^i 4^{\mathfrak{g}+1}$ **then**
      $\mathcal{S}$ = Halve $(\mathbf{k}, \mathcal{S}_i)$ // coreset of size $2^{i-1}4^{\mathfrak{g}+1}$
      $\mathcal{S}_{i+1} \leftarrow \mathcal{S}_{i+1} \cup \mathcal{S}$
      $\mathcal{S}_i \leftarrow \{\}$
    end
  end
  return $\mathcal{S}_{\lfloor \log_4 t \rfloor + 1}$ // Coreset stored in the highest level is the desired approximation at current stage
end

Next, we analyze the space complexity of the streaming version of the algorithm. The intuition for gains in storage is very similar to that for running time: we only maintain and run Halve only on subsets of much smaller size than the original input. See App. G for the proof.

Proposition 1 (Compress Streaming Memory Bound) *Let Halve be an algorithm that that takes as input a point sequence $\mathcal{S}_{\text{in}}$ of size n and return a data set $\mathcal{S}$ with size $n/2$ with space complexity $s_{\text{Halve}}(n)$. Then, at any time t, streaming version of* Compress *with oversampling factor $\mathfrak{g}$ has space complexity*

$$s_{\text{CP}}(t) = 4^{g+3}\sqrt{t}d + s(2^{\mathfrak{g}+1}\sqrt{t})d$$

real words of memory.

G.1 Proof of Prop. 1: Compress Streaming Memory Bound

For convenience assume that n is divisible by $4^{\mathfrak{g}+1}$. 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 $2^{i+2}4^{\mathfrak{g}} - 1$ 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 memory used up to time t is bounded by

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

data points stored. 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 $s\left(\sqrt{t}2^{\mathfrak{g}+1}\right)$ additional points. Each data point is stored with d real words of memory giving the required result.

H GUARANTEES FOR RECHALVE

Algorithm 4: Recursive Halving (**RecHalve**)

Input: Input point sequence $\mathcal{S}_{\text{in}}$ of size n, Halving algorithm HALVE, Thinning factor m
if $m = 0$ **then return** $\mathcal{S}_{\text{in}}$;
else
  $\mathcal{S}_1 \leftarrow \text{HALVE}(\mathcal{S}_{\text{in}})$ // halved coreset size $n/2$
  return *RecHalve* $(\mathcal{S}_1, \text{HALVE}, m-1)$
end

Proposition 2 (RecHalve Subgamma Guarantees) *Let* HALVE *be a halving algorithm that is* $\Gamma_+^{\mathcal{F}}\left(\sigma^2_{\text{HALVE}}, c_{\text{HALVE}}\right)$*. Then,* ***RecHalve*** *with depth parameter* m *is a* $2^{-m}\sqrt{n}$*-root thinning algorithm and is* $\Gamma_+^{\mathcal{F}}$ *with parameters*

$$\sigma^2_{RH}(n,m) = \textstyle\sum_{i=0}^{m-1} 4^i \sigma^2_{\text{HALVE}}\left(2^{-i}n\right), \quad \textit{and}$$
$$c_{RH}(n,m) = \max_{0\leq i\leq m-1}\left(2^i \cdot c_{\text{HALVE}}\left(n2^{-i}\right)\right).$$

Denote by $\mathcal{S}_1$ the output of the halving algorithm after the first round. Then,

$$\psi_{\text{RH}}(\mathcal{S}_{\text{in}}; m) = \psi_{\text{HALVE}}(\mathcal{S}_{\text{in}}) + 2\psi_{\text{RH}}(\mathcal{S}_1; m-1).$$

From the $\Gamma_+^{\mathcal{F}}$ guarantees of HALVE and noting the independence conditioned on the inputs, we have

$$\sigma^2_{\text{RH}}(n,m) = \sigma^2_{\text{HALVE}}(n) + 4\sigma^2_{\text{RH}}(n/2, m-1)$$
$$c_{\text{RH}}(n,m) = \max\left(c_{\text{HALVE}}\left(n\right), 2\cdot c_{\text{RH}}\left(n/2, m-1\right)\right)$$

Also note that $\sigma^2_{\text{RH}}(n,0) = c_{\text{RH}}(n,0) = 0$. Unrolling the recursion, we get

$$\sigma^2_{\text{RH}}(n,m) = \textstyle\sum_{i=0}^{m-1} 4^i \sigma^2_{\text{HALVE}}\left(2^{-i}n\right)$$
$$c_{\text{RH}}(n,m) = \max_{0\leq i\leq m-1}\left(2^i \cdot c_{\text{HALVE}}\left(n2^{-i}\right)\right)$$

as required.

Proposition 3 (RecHalve Running Time) *Let* HALVE *be a halving algorithm with running time* $h(n)$ *on inputs of size* n*. Then,* ***RecHalve*** *with depth parameter* m *is a* $2^{-m}\sqrt{n}$*-root algorithm with running time*

$$\textstyle\sum_{i=0}^{m-1} h(n2^{-i})$$

on inputs of size n*.*

I PROOF OF THEOREM 3: SUB-GAMMA PROPERTY AND RUNTIME OF COMPRESS++

As in the definition of the algorithm, let $\mathcal{S}_{\text{COMPRESS}}$ denote the output of COMPRESS. Recall that from Proposition 1, we have that

$$\psi_{\text{COMPRESS}}(\mathcal{S}_{\text{in}}; \mathfrak{g}) \triangleq \sum_{x \in \mathcal{S}_{\text{in}}} \mathbf{k}(x, \cdot) - \frac{\sqrt{n}}{2^{\mathfrak{g}}} \sum_{x \in \mathcal{S}_{\text{COMPRESS}}} \mathbf{k}(x, \cdot)$$

is Γ_+^f-sub-Gaussian with parameter

$$\sigma_{\text{COMPRESS}}(n, \mathfrak{g}) = \sqrt{n 4^{-\mathfrak{g}-1} \sum_{i=0}^{\log_4 n - g} \sigma^2_{\text{THIN}}\big(2^{\mathfrak{g}+1-i}\sqrt{n}, 1/2\big)}$$
$$c_{\text{COMPRESS}} = \max_{0 \le i \le \log_4 n - g - 1} \Big(\sqrt{n} 2^{-\mathfrak{g}-i-1} c_{\text{THIN}}\big(\sqrt{n} 2^{\mathfrak{g}+1-i}, 1/2\big)\Big).$$

Furthermore, by assumption, we have that

$$\psi_{\text{THIN}}(\mathcal{S}_{\text{COMPRESS}}) \triangleq \sum_{x \in \mathcal{S}_{\text{COMPRESS}}} \mathbf{k}(x, \cdot) - 2^{\mathfrak{g}} \sum_{x \in \mathcal{S}_{\text{COMPRESS++}}} \mathbf{k}(x, \cdot)$$

is Γ_+^f with parameters

$$\sigma^2_{\text{THIN}}\big(2^{\mathfrak{g}}\sqrt{n}, 2^{-\mathfrak{g}}\big)$$

and

$$c_{\text{THIN}}\big(2^{\mathfrak{g}}\sqrt{n}, 2^{-\mathfrak{g}}\big)$$

conditioned on the input $\mathcal{S}_{\text{COMPRESS}}$. Also, note that

$$\psi_{\text{COMPRESS++}} = \psi_{\text{COMPRESS}} + \frac{\sqrt{n}}{2^{\mathfrak{g}}} \psi_{\text{THIN}}(\mathcal{S}_{\text{COMPRESS}})$$

Thus, we have that $\psi_{\text{COMPRESS++}}$ is Γ_+^f with parameters

$$\sigma^2_{\text{THIN}}(n, \mathfrak{g}) = n 4^{-\mathfrak{g}-1} \sum_{i=0}^{\log_4 n - g} \sigma^2_{\text{THIN}}\big(2^{\mathfrak{g}+1-i}\sqrt{n}, 1/2\big) + n 4^{-\mathfrak{g}} \sigma^2_{\text{THIN}}\big(2^{\mathfrak{g}}\sqrt{n}, 2^{-\mathfrak{g}}\big)$$

and

$$c_{\text{CP++}}(n, \mathfrak{g}) = \max\Big(\sqrt{n} 2^{-\mathfrak{g}} c_{\text{THIN}}\big(2^{\mathfrak{g}}\sqrt{n}, 2^{-\mathfrak{g}}\big), \max_{0 \le i \le \log_4 n - g - 1}\Big(\sqrt{n} 2^{-\mathfrak{g}-i-1} c_{\text{CP++}}\big(\sqrt{n} 2^{\mathfrak{g}+1-i}, 1/2\big)\Big)\Big)$$

as required.

The runnning time bound follows from the running time of $\text{COMPRESS}\big(\text{THIN}(1/2), \mathfrak{g}\big)$ and $\text{THIN}\big(2^{\mathfrak{g}}\sqrt{n}, 2^{-\mathfrak{g}}\big)$.

J COMPRESS++ CALCULATIONS

The squared sub-Gaussian constant of KT with input size $\sqrt{n} \cdot 2^{\mathfrak{g}}$ and output size $\sqrt{n}$ is

$$\sigma^2_{\textbf{KT}}(\sqrt{n} 2^{\mathfrak{g}}; \sqrt{n}) = C \cdot \frac{4^{\mathfrak{g}} - 1}{3} \log_4(\sqrt{n} 2^{\mathfrak{g}})$$

where C is the constant floating with kernel-halving. The run-time is $C' n 4^{\mathfrak{g}}$. Thus if we run COMPRESS ($\mathfrak{g}$) with kernel halving, and then KT on the output of size $\sqrt{n} \cdot 2^{\mathfrak{g}}$, the overall sub-Gaussian constant for

$$\psi_{\textbf{Cg-KT}}(\mathcal{S}_{\text{in}}) \triangleq \psi_{\text{CP}}(\mathcal{S}_{\text{in}}; \mathfrak{g}) + \frac{\sqrt{n}}{2^{\mathfrak{g}}} \psi_{\textbf{KT}}(\mathcal{S}_{\text{CP}})$$

is given by

$$\begin{aligned} &C \frac{n}{4^{\mathfrak{g}+1}} \cdot \frac{1}{4}(\log_4 n + 3\mathfrak{g} + 3)(\log_4 n - \mathfrak{g}) + \frac{n}{4^{\mathfrak{g}}} \cdot C \cdot \frac{4^{\mathfrak{g}} - 1}{3} \cdot \log_4(\sqrt{n} 2^{\mathfrak{g}}) \\ &= C \frac{n}{4^{\mathfrak{g}+1}} \cdot \frac{1}{4}(\log_4 n + 3\mathfrak{g} + 3)(\log_4 n - \mathfrak{g}) + \frac{n}{4^{\mathfrak{g}}} \cdot C \cdot \frac{4^{\mathfrak{g}} - 1}{6} \cdot (\log_4 n + \mathfrak{g}). \end{aligned}$$

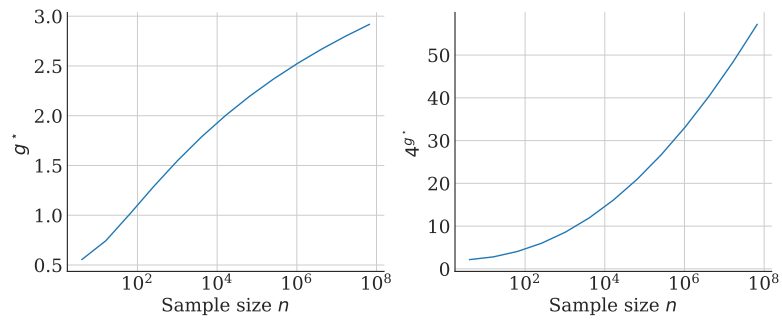

Figure 4: Scaling of optimal hyper-parameter for Compress-KT.

Define $\alpha \triangleq \log_4 n$ and $a \triangleq 4^{\mathfrak{g}}$, and note that $\log_4 x = c \log x$ for $c \triangleq 1/\log 4$. Then the expression simplifies to

$$\begin{aligned}
&\tfrac{n}{2}\Big[\tfrac{1}{8}\cdot\tfrac{1}{4^{\mathfrak{g}}}\big(\alpha^2-\alpha\mathfrak{g}+3\alpha\mathfrak{g}-3\mathfrak{g}^2+3\alpha-3\mathfrak{g}\big)+\tfrac{1}{3}\cdot\tfrac{1}{4^{\mathfrak{g}}}\big((4^{\mathfrak{g}}-1)\alpha+(4^{\mathfrak{g}}-1)\mathfrak{g}\big)\Big]\\
&=\tfrac{n}{2}\Big[\tfrac{1}{8}\cdot\tfrac{1}{4^{\mathfrak{g}}}\big(\alpha^2+3\alpha+\mathfrak{g}(2\alpha-3)-3\mathfrak{g}^2\big)+\tfrac{1}{3}\cdot\tfrac{1}{4^{\mathfrak{g}}}\big((4^{\mathfrak{g}}-1)\alpha+(4^{\mathfrak{g}}-1)\mathfrak{g}\big)\Big]\\
&=\tfrac{n}{2a}\Big[\tfrac{1}{8}\big(\alpha^2+3\alpha+c\log a(2\alpha-3)-3c^2(\log a)^2\big)+\tfrac{1}{3}(a-1)(\alpha+c\log a)\Big].
\end{aligned}$$

Note that $\frac{d}{da}[g(a)/a]=0 \Leftrightarrow g(a)=ag'(a)$. Taking the derivative and setting it to 0, we have

$$\begin{aligned}
&3\alpha^2+9\alpha+3c(2\alpha-3)\log a-9c^2(\log a)^2+8a(\alpha+c\log a)-8(\alpha+c\log a)\\
&\qquad\qquad = 3c(2\alpha-3)-18c^2\log a+8a(\alpha+c\log a)+8ac-8c
\end{aligned}$$

which implies

$$\begin{aligned}
&8ac = 3\alpha^2+9\alpha+3c(2\alpha-3)\log a-9c^2(\log a)^2-8(\alpha+c\log a)-3c(2\alpha-3)+18c^2\log a+8c\\
&\Rightarrow a = \tfrac{1}{8c}\big[3\alpha^2+\alpha+17c-6c\alpha+(6c\alpha-17c+18c^2)\log a-9c^2(\log a)^2\big]\\
&\quad = \tfrac{17-6\alpha}{8}+\tfrac{3\alpha^2+\alpha}{8c}+\tfrac{(6\alpha-17+18c)}{8}\log a-\tfrac{9c}{8}(\log a)^2. \qquad (9)
\end{aligned}$$

Let g_{opt} denote the optimal g obtained by solving (9) (recall $4^g = a$). In Fig. 4, we plot the values computed numerically using fixed point iterations for (9). We note that even for large sample values of $n = 10^7$, a default choice of $g_{opt} = 3$ suffices.

Also, note that from the above expression We highlight that with such a choice of g, the run-time (using Theorem 4) is bounded as

$$\begin{aligned}
\tfrac{n}{4^{\mathfrak{g}}}+C(\log_4 n-\mathfrak{g})4^{\mathfrak{g}+1}n+Cn4^{\mathfrak{g}} &\le \tfrac{n}{\log_4 n}+Cn(4(\log_4 n)^2+\log_4 n-4\log_4 n\log_4\log_4 n)\\
&\le 5Cn(\log_4 n)^2 \quad \text{with} \quad \mathfrak{g}=\mathfrak{g}_{\text{opt}}.
\end{aligned}$$

K Proof of Theorem 4: Compress++ MMD Guarantees

Recall that

$$\psi_{\text{CP++}} = \psi_{\text{CP}} + \tfrac{\sqrt{n}}{2^{\mathfrak{g}}}\psi_{\text{THIN}}(\mathcal{S}_{\text{CP}})$$

By assumption, we have that on input $\mathcal{S}_{\text{in}}$ of size n, THIN outputs $\mathcal{S}_{\text{out}}$ such that

$$\Pr\Big[\text{MMD}(\mathcal{S}_{\text{in}},\mathcal{S}_{\text{out}}) > a_{\ell,\alpha}+\sqrt{2v_{\ell,\alpha}t}+c_{\ell,\alpha}t\Big] \le e^{-t}.$$

Again, recall that $n\,\text{MMD} = \|\psi\|_{\mathbf{k}}$. Thus, we need to bound

$$\begin{aligned}
\tfrac{1}{n}\|\psi_{\text{CP++}}\|_{\mathbf{k}} &= \tfrac{1}{n}\|\psi_{\text{CP}}+\tfrac{\sqrt{n}}{2^{\mathfrak{g}}}\psi_{\text{THIN}}(\mathcal{S}_{\text{CP}})\|_{\mathbf{k}}\\
&\le \tfrac{1}{n}\|\psi_{\text{CP}}\|_{\mathbf{k}}+\tfrac{1}{\sqrt{n}2^{\mathfrak{g}}}\|\psi_{\text{THIN}}\|_{\mathbf{k}}\\
&= \text{MMD}(\mathcal{S}_{\text{in}},\mathcal{S}_{\text{CP}})+\text{MMD}(\mathcal{S}_{\text{CP}},\mathcal{S}_{\text{out}}).
\end{aligned}$$

From Theorem 2, we have that

$$\Pr\left[\mathrm{MMD}(\mathcal{S}_{\mathrm{in}}, \mathcal{S}_{\mathrm{CP}}) > 2\gamma R_{\mathrm{MMD,CP}} + \sqrt{2\gamma\sigma^2_{\mathrm{MMD,CP}}}\right] \leq 2(n+1)\exp(-\gamma)$$

$$\text{for} \quad \sigma^2_{\mathrm{MMD,CP}} \leq 2^{-2g-2}\left(\sqrt{8v_{\sqrt{n}2^{\mathfrak{g}+1}} + 2a^2_{\sqrt{n}2^{\mathfrak{g}+1}}} + 4c_{\sqrt{n}2^{\mathfrak{g}+1}}\right)^2(\log_4 n - \mathfrak{g}),$$

$$\text{and} \quad R_{\mathrm{MMD,CP}} \leq 2^{-\mathfrak{g}-1}\left(\sqrt{8v_{\sqrt{n}2^{\mathfrak{g}+1}} + 2a^2_{\sqrt{n}2^{\mathfrak{g}+1}}} + 4c_{\sqrt{n}2^{\mathfrak{g}+1}}\right).$$

Since, the guarantees of THINare conditioned on the inputs, we get

$$\begin{aligned}
&\Pr\left[\mathrm{MMD}(\mathcal{S}_{\mathrm{in}}, \mathcal{S}_{\mathrm{CP++}}) > 2tR_{\mathrm{MMD,CP}} + \sqrt{2t\sigma^2_{\mathrm{MMD,CP}}} + a_{2^{\mathfrak{g}}\sqrt{n},2^{\mathfrak{g}}} + \sqrt{2v_{2^{\mathfrak{g}}\sqrt{n},2^{\mathfrak{g}}}t} + c_{2^{\mathfrak{g}}\sqrt{n},2^{-\mathfrak{g}}}t\right] \leq \\
&\Pr\left[\mathrm{MMD}(\mathcal{S}_{\mathrm{in}}, \mathcal{S}_{\mathrm{CP}}) > 2tR_{\mathrm{MMD,CP}} + \sqrt{2t\sigma^2_{\mathrm{MMD,CP}}}\right] + \Pr\left[\mathrm{MMD}(\mathcal{S}_{\mathrm{in}}, \mathcal{S}_{\mathrm{out}}) > a_{2^{\mathfrak{g}}\sqrt{n},2^{\mathfrak{g}}} + \sqrt{2v_{2^{\mathfrak{g}}\sqrt{n},2^{\mathfrak{g}}}t} + c_{2^{\mathfrak{g}}\sqrt{n},2^{-\mathfrak{g}}}t\right] \\
&\leq \left(e^{-t}\right) + \left(2(n+1)e^{-t}\right) \\
&\leq 2(n+2)e^{-t}.
\end{aligned}$$

Thus, we get

$$\Pr\left[\mathrm{MMD}(\mathcal{S}_{\mathrm{in}}, \mathcal{S}_{\mathrm{CP++}}) \geq a_{2^{\mathfrak{g}}\sqrt{n},2^{\mathfrak{g}}} + \sqrt{2(\sigma^2_{\mathrm{MMD,CP}} + v_{2^{\mathfrak{g}}\sqrt{n},2^{\mathfrak{g}}})t} + (c_{2^{\mathfrak{g}}\sqrt{n},2^{\mathfrak{g}}} + 2R_{\mathrm{MMD,CP}})t\right] \leq 2(n+2)e^{-t}$$

as required.

L ADDITIONAL EXPERIMENTS. AND SUPPLEMENTARY INFORMATION

MoG Target Our mixture of Gaussians target is given by $\mathbb{P} = \frac{1}{M}\sum_{j=1}^{M}\mathcal{N}(\mu_j, \mathbf{I}_d)$ for $M \in \{4, 6, 8\}$ where

$$\begin{aligned}
&\mu_1 = [-3, 3]^\top, \quad \mu_2 = [-3, 3]^\top, \quad \mu_3 = [-3, -3]^\top, \quad \mu_4 = [3, -3]^\top, \\
&\mu_5 = [0, 6]^\top, \quad \mu_6 = [-6, 0]^\top, \quad \mu_7 = [6, 0]^\top, \quad \mu_8 = [0, -6]^\top.
\end{aligned}$$

Visualizing coresets In Fig. 5, we visualize the coresets for a 32-MoG target in with means located on two concntric circles, from where we observe the visible advantages of all the thinning strategies over standard thinning, and the comparable performance of the COMPRESS++ procedures. We see that even with 32 points all procedures succeed in locating the means for the MoG, thereby providing a better approximation to the underlying target with i.i.d. points.

In Figs. 6 and 7, we present the MMD errors of KT and herding experiments for the MOG target, and notice again that COMPRESS++ provides a competitive performance to the original algorithm.

MCMC Target Our set-up for MCMC experiments follows closely the set-up of Dwivedi & Mackey (2021b). For complete details on the targets and sampling algorithms we refer the reader to Riabiz et al. (2020a, Sec. 4).

Goodwin and Lotka-Volterra experiments The dataset of Riabiz et al. (2020b) consists of output of four distinct MCMC procedures targeting each of two $d = 4$-dimensional posterior distributions $\mathbb{P}$: (1) a posterior over the parameters of the *Goodwin model* of oscillatory enzymatic control (Goodwin, 1965), and (2) a posterior for the *Lotka-Volterra model* of oscillatory predator-prey evolution (Lotka, 1925; Volterra, 1926)—both of these are differential equation based models and often used as benchmark datasets for Bayesian infere.ce Riabiz et al. (2020b) provide 2×10^6 sample points for each task from four MCMC chains: Gaussian random walk (RW), adaptive Gaussian random walk (adaRW, Haario et al., 1999), Metropolis-adjusted Langevin algorithm (MALA, Roberts & Tweedie, 1996), and pre-conditioned MALA (pMALA, Girolami & Calderhead, 2011).

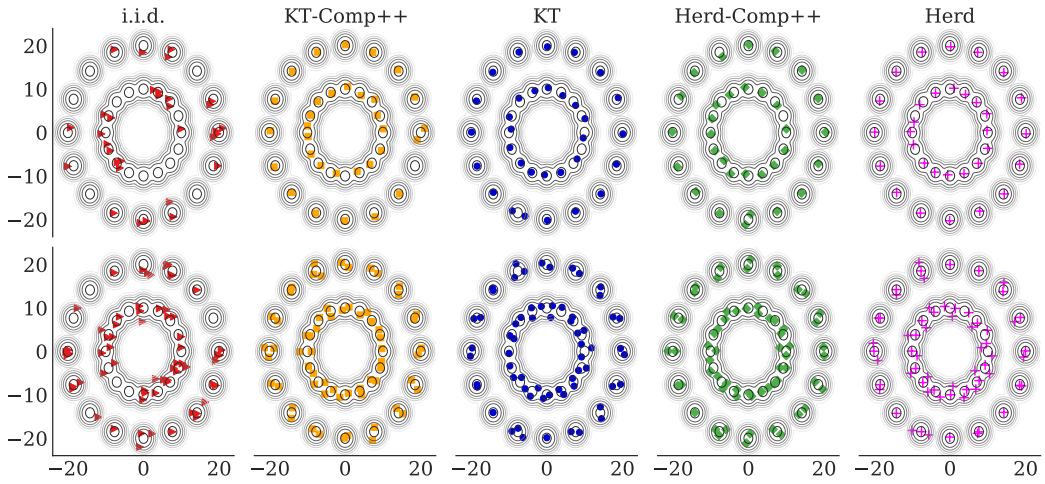

Figure 5: Scatter plot of various coresets for 32-MoG target.

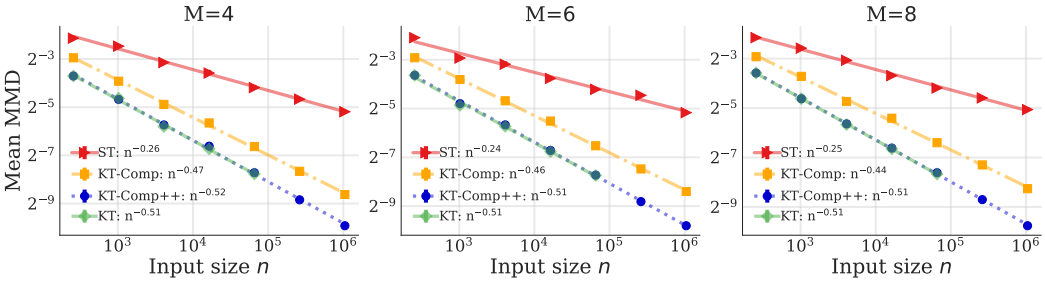

Figure 6: MoG target experiments with KT and its COMPRESS and COMPRESS++ variants

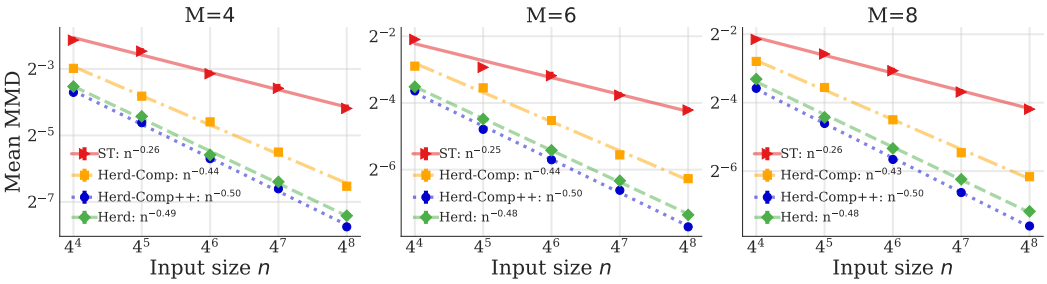

Figure 7: MoG target with kernel herding and its COMPRESS and COMPRESS++ variants

Hinch experiments Riabiz et al. (2020b) also made available the output of two independent Gaussian random walk MCMC chains targeting each of two $d = 38$-dimensional posterior distributions $\mathbb{P}$ for a cardiac experiment that approximates a posterior over the parameters of the Hinch model of calcium signalling in cardiac cells (Hinch et al., 2004) and (2) a tempered version of the same posterior, as defined by Riabiz et al. (2020a, App. S5.4).

Burn-in times For all chains, the initial burn-in points of each chain using the maximum burn-in period reported in Riabiz et al. (2020a, Tabs. S4 & S6, App. S5.4). For each Hinch chain, we further normalized by subtracting the post-burn-in sample mean and dividing each coordinate by its post-burn-in sample standard deviation. To obtain an input sequence $\mathcal{S}_{\text{in}}$ of length n to be fed into a thinning algorithm, we downsample the remaining even indices of points using standard thinning (odd indices are held out). When applying standard thinning to any Markov chain output, we adopt the convention of keeping the final sample point.

The selected burn-in periods for the Goodwin task were 820,000 for RW; 824,000 for adaRW; 1,615,000 for MALA; and 1,475,000 for pMALA. The respective numbers for the Lotka-Volterra task were 1,512,000 for RW; 1,797,000 for adaRW; 1,573,000 for MALA; and 1,251,000 for pMALA.

MCMC Results From Fig. 8, we note that in some cases standard thinning itself provides a better than Monte Carlo approximation to the input, but KT and its COMPRESS++ variant are able to provide further improvement. For the Hinch experiments, we found that the symmetrization step as noted in Rem. 4 (and enforced in all the KT/Herding experiments) lead to worse performance, and not implementing it enabled COMPRESS++ (labeled as KT-Comp++(NS)) to once again provide competitive performance to KT.

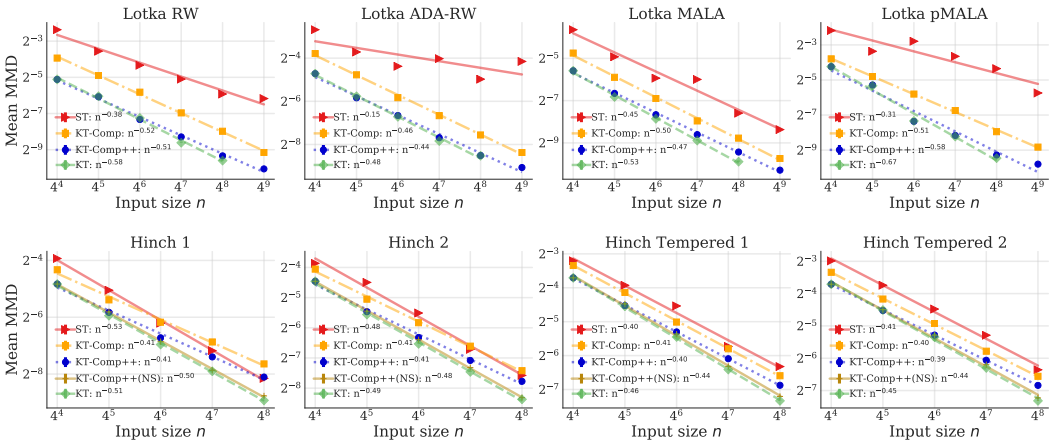

Figure 8: MCMC Target Experiments with KT and its COMPRESS and COMPRESS++ variants.