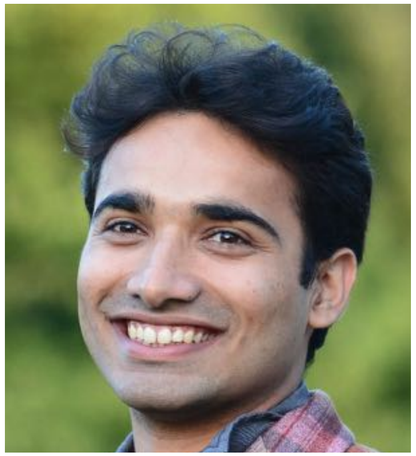
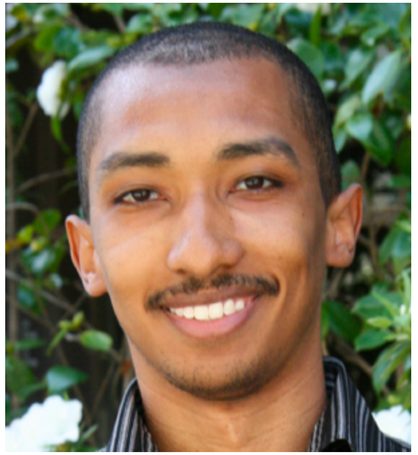
Near-Optimal Compression In Near-Linear Time

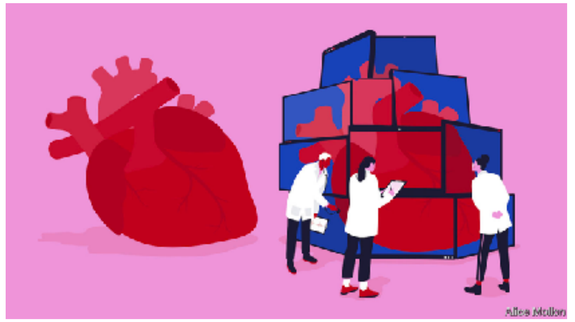
Raaz Dwivedi, Lester Mackey, Abhishek Shetty
raaz@mit.edu


MIT EECS | Harvard John A. Paulson School of Engineering and Applied Sciences | Microsoft Research | Berkeley EECS

MIT LIDS Student Conference
Jan 26, 2022

Computational cardiology

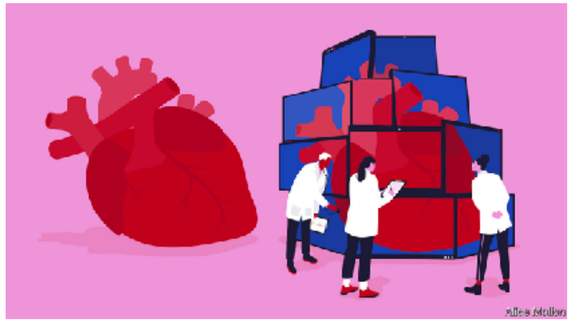


- Modeling **digital twin heart** to predict therapy response in a *non-invasive way* requires single-cell modeling.
  Commonly used strategy:
    - Estimate single cell model using Bayesian set-up: Use millions of Markov chain Monte Carlo (MCMC) points to approximate posterior $\mathbb{P}^\star$
    - Feed these samples to whole-heart simulator

$$\mathbb{P}^\star f \triangleq \int f(x) d\mathbb{P}^\star(x) \approx \frac{1}{n} \sum_{i=1}^{n} f(x_i) \triangleq \mathbb{P}_n f$$

x_i = MCMC sample for single cell model parameters
f = heart simulator

Computational cardiology



- Modeling **digital twin heart** to predict therapy response in a *non-invasive way* requires single-cell modeling.
  Commonly used strategy:
  - Estimate single cell model using Bayesian set-up: Use millions of Markov chain Monte Carlo (MCMC) points to approximate posterior $\mathbb{P}^\star$
  - Feed these samples to whole-heart simulator

$$\mathbb{P}^\star f \triangleq \int f(x) d\mathbb{P}^\star(x) \approx \frac{1}{n} \sum_{i=1}^{n} f(x_i) \triangleq \mathbb{P}_n f$$

x_i = MCMC sample for single cell model parameters
f = heart simulator

1 Million MCMC samples ~ 2 weeks
Single evaluation of f ~ 5 weeks

Can NOT use all million samples….!

Goal: Represent $\mathbb{P}^\star$ using a few high quality points $(x_i)_{i=1}^n$

- Typical solutions like **I.I.D. sampling, and MCMC sampling** exhibit the **slow root-n Monte Carlo rate** $\left|\mathbb{P}^\star f - \mathbb{P}_n f\right| = \Theta(n^{-1/2})$, e.g., around a million points for 0.1% error

- Computationally prohibitive for expensive f, and common fixes uniform thinning, or **standard thinning—-choose every t—th point**

- **Accuracy degrades with such thinning**—$\Theta(\sqrt{t/n})$ worst-case error—same as the Monte Carlo rate with n/t points;
  e.g., $n^{-1/4}$ rate with $\sqrt{n}$ points

Can we do better?

Minimax lower bounds

- There exists some $\mathbb{P}^\star$ such that the worst-case integration error
  - Is $\Omega(n^{-1/2})$ for any compression scheme returning $\sqrt{n}$ points
    [Philips and Tai, 2020]

Minimax lower bounds

- There exists some $\mathbb{P}^\star$ such that the worst-case integration error
  - Is $\Omega(n^{-1/2})$ for any compression scheme returning $\sqrt{n}$ points
    [Philips and Tai, 2020]
  - Is $\Omega(n^{-1/2})$ for any approximation based on $\sqrt{n}$ i.i.d. points
    [Tolstikhin, Sriperumbudur, and Muandet, 2017]

This talk: Kernel thinning-Compress++

- **KT-Compress++**: A practical strategy based on two new algorithms to provide near-optimal compression in near-linear time
    - **Kernel thinning (KT)**: Provides near-optimal compression
    - **Compress++**: Provides significantly reduced runtime for generic thinning algorithms with minimal worsening of error

This talk: Kernel thinning-Compress++

- **KT-Compress++**: A practical strategy based on two new algorithms to provide near-optimal compression in near-linear time
    - **Kernel thinning (KT)**: Provides near-optimal compression
    - **Compress++**: Provides significantly reduced runtime for generic thinning algorithms with minimal worsening of error
- Overall, KT-Compress++ a solution for finding
    - better than Monte Carlo points
    - high quality coresets
    - good prototypes

Problem set-up

- **Input**:

  Points $(x_i)_{i=1}^n \subset \mathbb{R}^d$ with empirical distribution $\mathbb{P}_{in} \triangleq \frac{1}{n}\sum_{i=1}^{n} \delta_{x_i}$

  Target output size s ($s = \sqrt{n}$ for heavy compression)

- **Goal:**

  Return a subset of input points with size s, empirical distribution $\mathbb{P}_{out}$ with error rate $o(s^{-1/2})$, i.e., better than Monte Carlo rate

Reproducing kernel Hilbert space (RKHS)

- RKHS of $\mathbf{k}$ is given by $\mathbb{H}_{\mathbf{k}} \triangleq \overline{\text{span}}\{\mathbf{k}(x, \cdot), x \in \mathbb{R}^d\}$
- $\mathbf{k}$ is universal when $\mathbb{H}_{\mathbf{k}}$ is dense in the space of continuous functions, e.g.,

$$\text{Gaussian } \mathbf{k}(x, y) = \exp\left(-\frac{1}{2}\|x - y\|^2\right); \text{ IMQ } \mathbf{k}(x, y) = \frac{1}{(1 + \|x - y\|_2^2)^{1/2}}$$

$\mathbf{k} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a reproducing kernel if the matrix $\mathbf{K} = (\mathbf{k}(x_i, x_j))_{i,j=1}^n$
is a symmetric positive definite matrix for any n and any $(x_1, \ldots, x_n)$

Kernel Thinning

$x_1, x_2, \ldots, x_n \in \mathbb{R}^d$

smooth decaying $\mathbf{k}$

$$\mathbb{P}_{in} \triangleq \frac{1}{n} \sum_{i=1}^{n} \delta_{x_i}$$

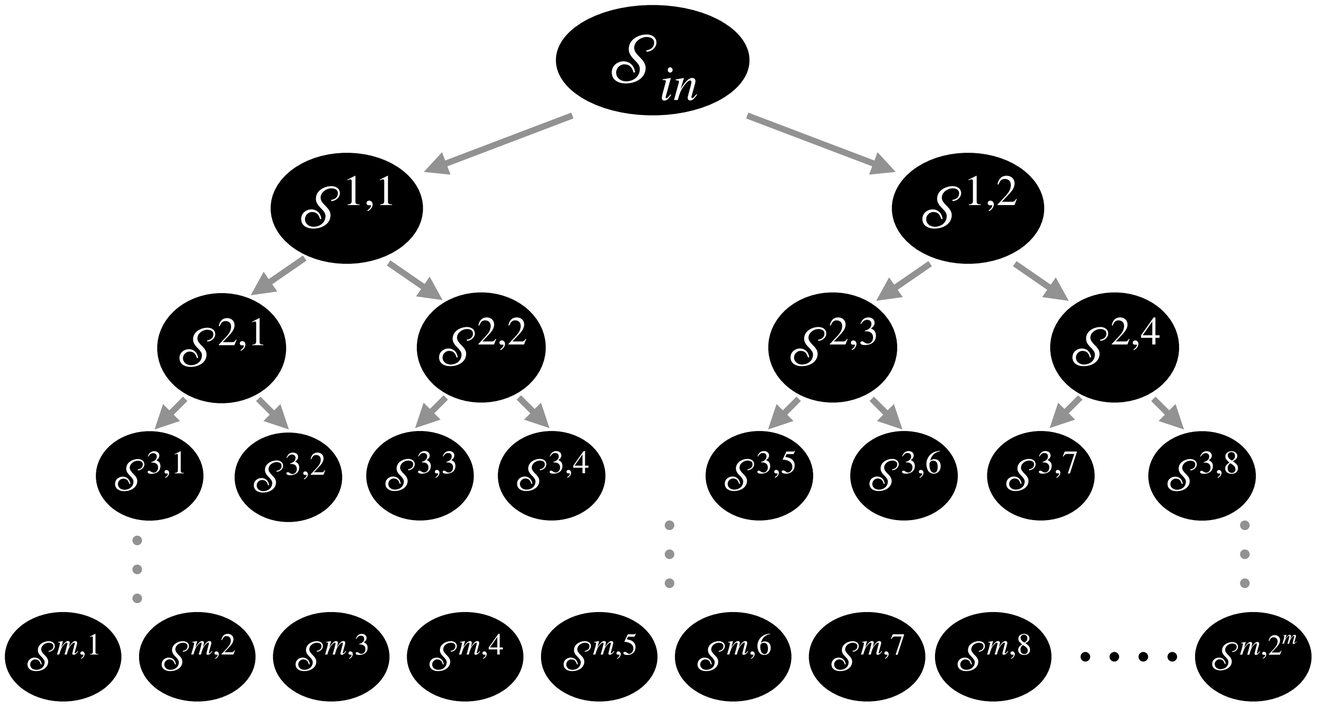
→ **Kernel Thinning (KT)** →

Non-uniform sub-sample of **size** s

$$y_1, \ldots, y_s$$

$$\mathbb{P}_{KT} \triangleq \frac{1}{s} \sum_{i=1}^{s} \delta_{y_i}$$

- **Stage 1:** Repeated rounds of **non-uniform splitting** the parent coreset in two equal-sized children coresets
- **Stage 2:** Point-by-point **refinement** of the best child coreset



Kernel Thinning: Better than Monte Carlo rate

$x_1, x_2, \ldots, x_n \in \mathbb{R}^d$

smooth decaying $\mathbf{k}$

$$\mathbb{P}_{in} \triangleq \frac{1}{n}\sum_{i=1}^{n} \delta_{x_i}$$

→ **Kernel Thinning (KT)** →

Non-uniform sub-sample of **size** s

$$y_1, \ldots, y_s$$

$$\mathbb{P}_{KT} \triangleq \frac{1}{s}\sum_{i=1}^{s} \delta_{y_i}$$

For any fixed $g \in \mathbb{H}_{\mathbf{k}}$, with high probability over the randomness in KT, we have

$$|\mathbb{P}_{in} g - \mathbb{P}_{KT} g| \lesssim \frac{1}{s} \cdot \|g\|_{\mathbf{k}} \sqrt{\|\mathbf{k}\|_\infty \left(\log s + \log\log(n/s)\right)}$$

Much faster than the Monte Carlo rate for standard/uniform thinning $\mathcal{O}\left(\frac{1}{\sqrt{s}}\right)$

Kernel Thinning: Better than Monte Carlo rate

$x_1, x_2, \ldots, x_n \in \mathbb{R}^d$

smooth decaying $\mathbf{k}$

$$\mathbb{P}_{in} \triangleq \frac{1}{n}\sum_{i=1}^{n} \delta_{x_i}$$

→ **Kernel Thinning (KT)** →

Non-uniform sub-sample of size $\sqrt{n}$

$$y_1, \ldots, y_{\sqrt{n}}$$

$$\mathbb{P}_{KT} \triangleq \frac{1}{\sqrt{n}}\sum_{i=1}^{\sqrt{n}} \delta_{y_i}$$

For any fixed $g \in \mathbb{H}_{\mathbf{k}}$, with high probability over the randomness in KT, we have

$$|\mathbb{P}_{in} g - \mathbb{P}_{KT} g| \precsim \frac{1}{\sqrt{n}} \cdot \|g\|_{\mathbf{k}} \sqrt{\|\mathbf{k}\|_\infty \log n}$$

Much faster than the Monte Carlo rate for standard/uniform thinning $\mathcal{O}\left(\frac{1}{n^{1/4}}\right)$

Kernel Thinning: Better than Monte Carlo rate

$x_1, x_2, \ldots, x_n \in \mathbb{R}^d$
smooth decaying $\mathbf{k}$
$\mathbb{P}_{in} \triangleq \frac{1}{n}\sum_{i=1}^{n} \delta_{x_i}$

→ **Kernel Thinning (KT)** →

Non-uniform sub-sample of size $\sqrt{n}$

$y_1, \ldots, y_{\sqrt{n}}$
$\mathbb{P}_{KT} \triangleq \frac{1}{\sqrt{n}}\sum_{i=1}^{\sqrt{n}} \delta_{y_i}$

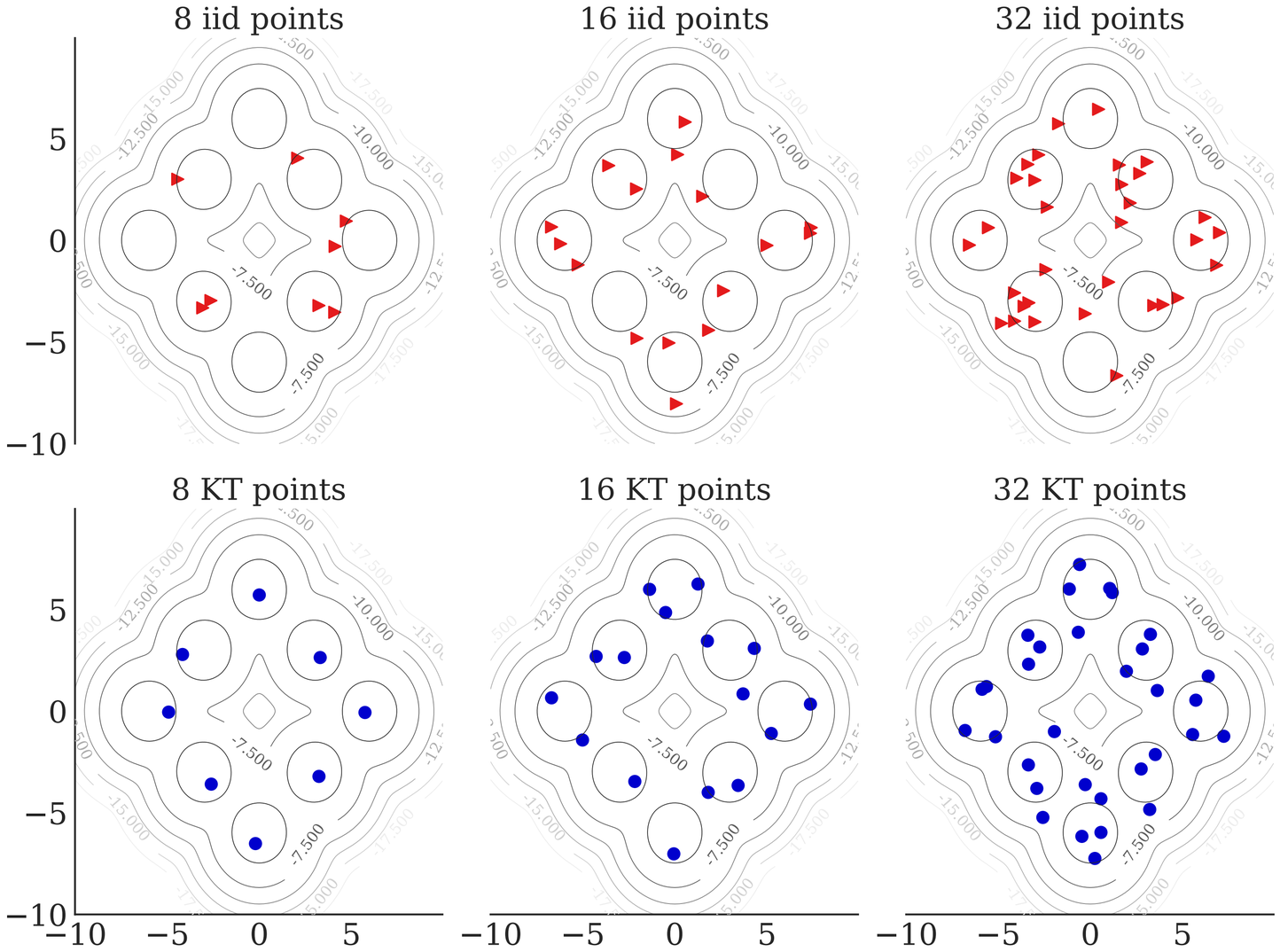
For any fixed $g \in \mathbb{H}_{\mathbf{k}}$, with high probability over the randomness in KT, we have

$$|\mathbb{P}_{in} g - \mathbb{P}_{KT} g| \precsim \frac{1}{\sqrt{n}} \cdot \|g\|_{\mathbf{k}} \sqrt{\|\mathbf{k}\|_\infty \log n}$$

$|\mathbb{P}^\star g - \mathbb{P}_{KT} g| = \mathcal{O}\left(\sqrt{\frac{\log n}{n}}\right)$ whenever $|\mathbb{P}^\star g - \mathbb{P}_{in} g| = \mathcal{O}\left(\sqrt{\frac{\log n}{n}}\right)$

Easily satisfied for input from iid sampling, MCMC, quadrature methods….

Kernel Thinning: Finds ``diverse'' set of points



Worst-case error: $\widetilde{\mathcal{O}}(n^{-1/2})$ with $\sqrt{n}$ points

- With high probability, the worst-case error—-MMD error—in the reproducing kernel Hilbert space (RKHS) satisfies

$$\sup_{\|g\|_{\mathbf{k}} \leq 1} |\mathbb{P}_{in} g - \mathbb{P}_{KT} g| \lesssim_d \begin{cases} n^{-1/2}\sqrt{\log n} & \text{(Compactly supported; e.g., B-spline } \mathbf{k}\text{)} \\ n^{-1/2}\sqrt{\log^{d/2+1} n \log\log n} & \text{(Sub-Gaussian tails; e.g., Gaussian } \mathbf{k}\text{)} \\ n^{-1/2}\sqrt{\log^{d+1} n \log\log n} & \text{(Sub-exponential tails; e.g., Matern } \mathbf{k}\text{)} \end{cases}$$

- For output size s, the MMD error is $\widetilde{\mathcal{O}}(1/s)$

Worst-case error: $\widetilde{\mathcal{O}}(n^{-1/2})$ with $\sqrt{n}$ points

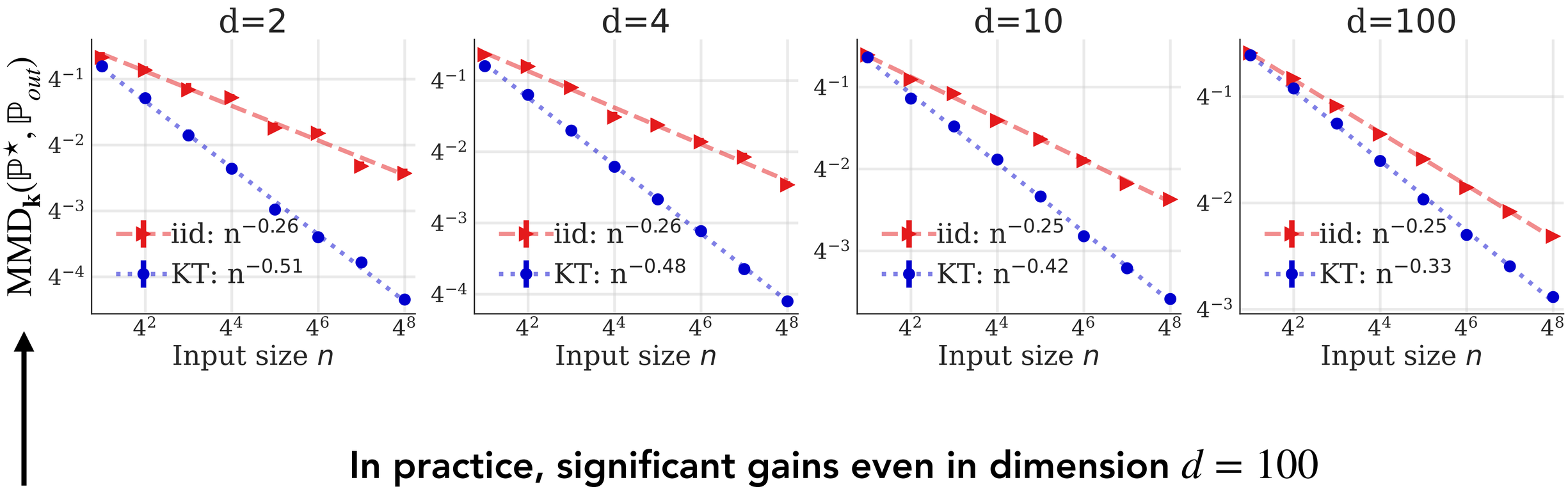
- With high probability, the worst-case error—-MMD error—in the reproducing kernel Hilbert space (RKHS) satisfies

$$\sup_{\|g\|_{\mathbf{k}}\leq 1} |\mathbb{P}^{\star} g - \mathbb{P}_{KT} g| \lesssim_d \begin{cases} n^{-1/2}\sqrt{\log n} & \text{(Compactly supported; e.g., B-spline } \mathbf{k}\text{)} \\ n^{-1/2}\sqrt{\log^{d/2+1} n \log\log n} & \text{(Sub-Gaussian tails; e.g., Gaussian } \mathbf{k}\text{)} \\ n^{-1/2}\sqrt{\log^{d+1} n \log\log n} & \text{(Sub-exponential tails; e.g., Matern } \mathbf{k}\text{)} \end{cases}$$

whenever $\sup_{\|g\|_{\mathbf{k}}\leq 1} |\mathbb{P}^{\star} g - \mathbb{P}_{in} g| \lesssim n^{-1/2}$ —-holds for iid / fast mixing MCMC input

KT vs iid: Gaussian $\mathbb{P}^\star$ in $\mathbb{R}^d$

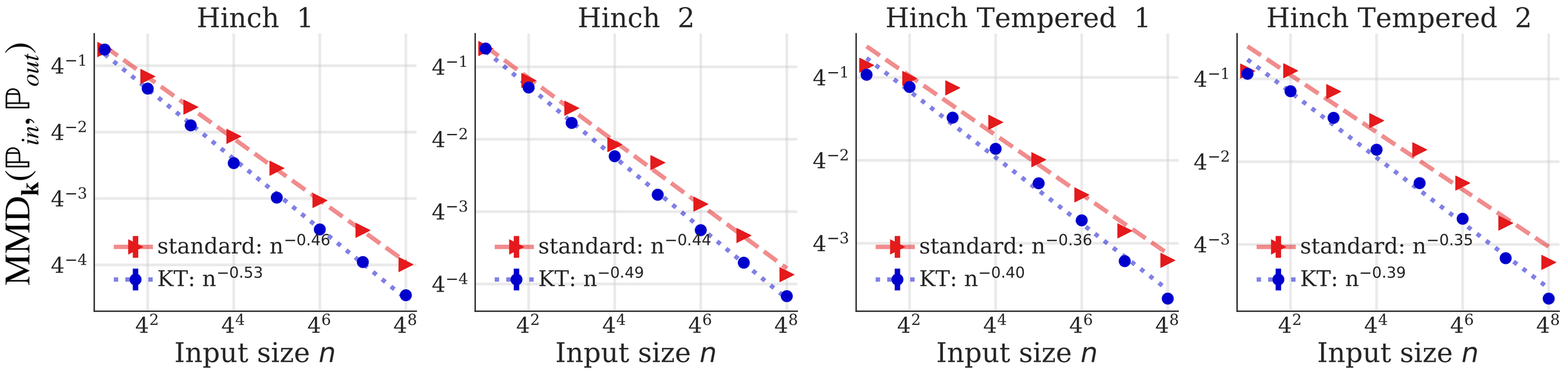
(Gaussian kernel with $\sigma^2 = 2d$)



In practice, significant gains even in dimension $d = 100$

(Worst-case error in the unit ball of Gaussian RKHS)

KT on MCMC samples from computational cardiology



In this setting with $d = 38$, standard thinning is already good,

but KT provides further improvement!

Note that each point saves 1000s of CPU hours!!

*MCMC samples taken from Riabiz-Chen-Cockayne-Swietach-Niederer-Mackey-Oates, 2021

Comparison with other works

- Ours is the **first result** that establishes **faster than Monte Carlo rate for generic $\mathbf{k}$ and $\mathbb{P}^\star$ with unbounded support**

- Prior results show better than Monte Carlo rates

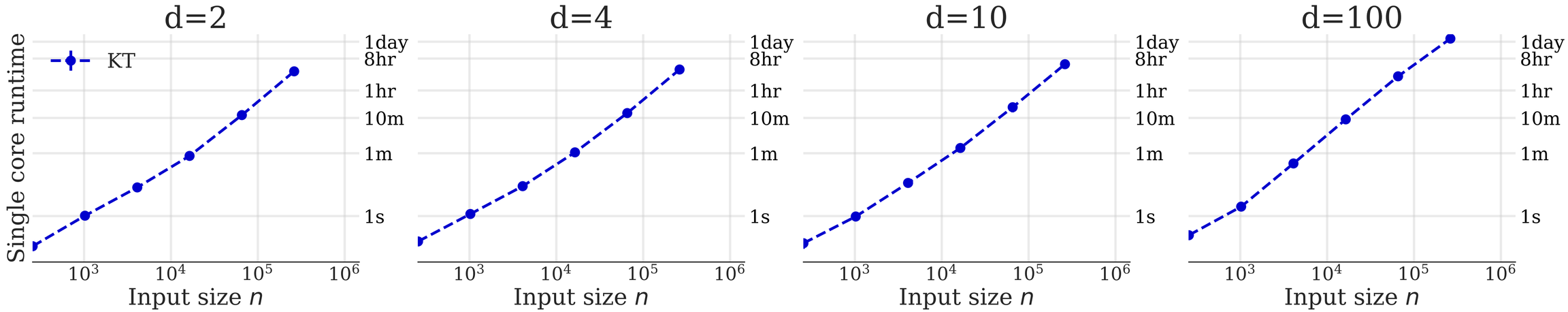
    - either empirically without a proof

    - or provide a proof under restrictive assumptions on $\mathbb{P}^\star$ or $\mathbf{k}$, e.g., uniform distribution on unit cube, or finite-dimensional $\mathbf{k}$

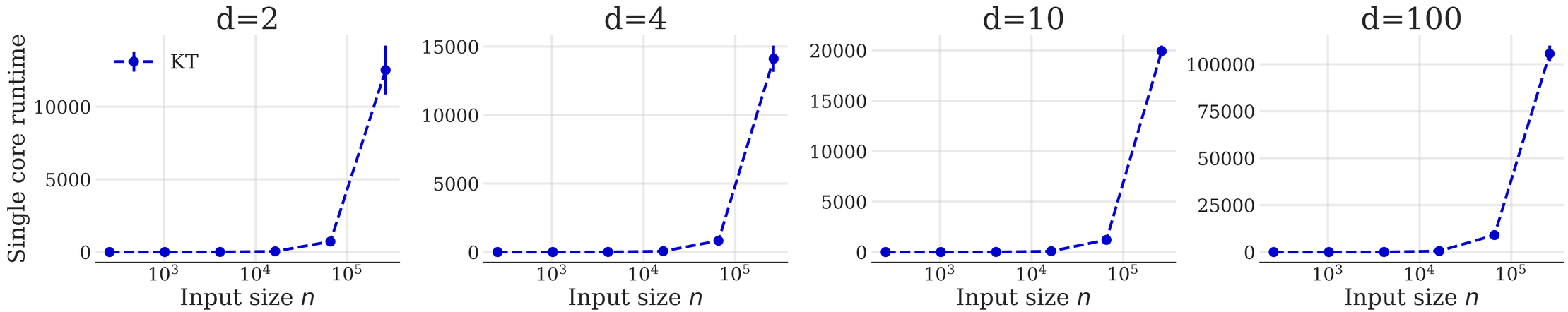
    - including Kernel herding, quasi Monte Carlo, Bayesian quadrature, support points, stein points, …

KT drawback: n^2 runtime with n input points

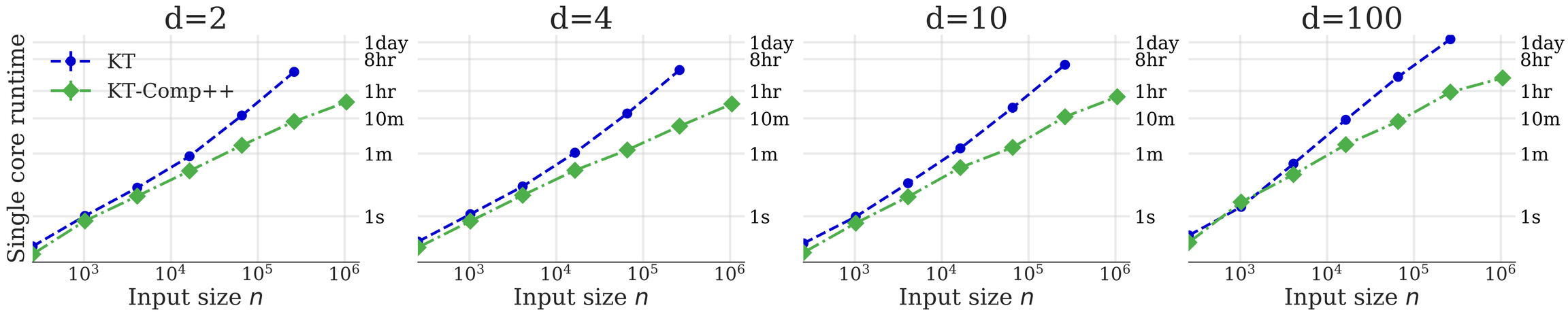
Y-axis = **Runtime in log-scale**



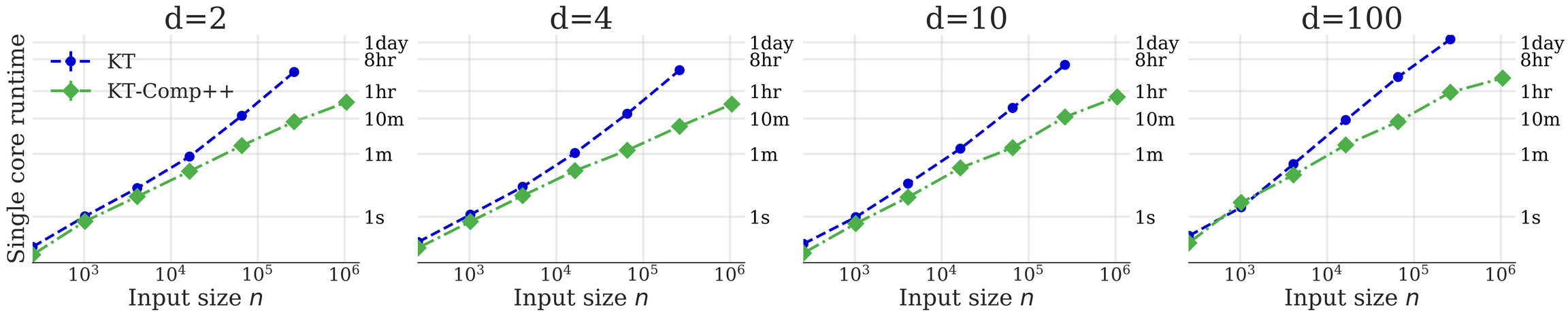
Y-axis = **Runtime in linear scale** (seconds)

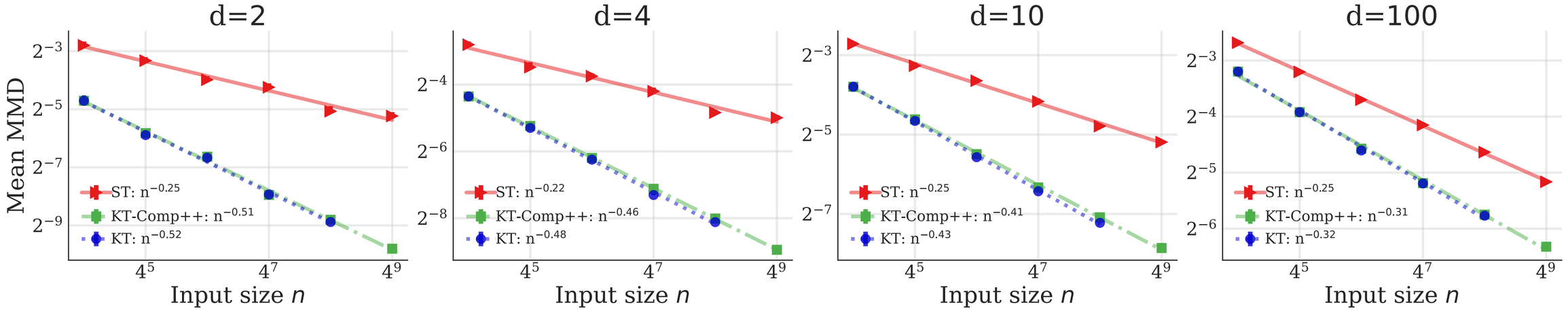


Compress++: Reducing runtime



Compress++: Reducing runtime with minimal loss in accuracy!!





Compress++: A recursive strategy to reduce runtime for **generic thinning** algorithms

Thinning algorithm with n^2 runtime and error $e(n)$ → Compress++ → $\sqrt{n}$-thinning algorithm with $n \log^3 n$ runtime and error $4e(n)$

- Compress($\mathcal{S}$, ALG):
  - If size($\mathcal{S}$) == 1, **Return** $\mathcal{S}$
  - Else:

    (i) Call Compress separately on 4 equal splits of $\mathcal{S}$

    (ii) Concatenate the 4 outputs from step (i)

    (iii) **Return** Halved output of step (ii) using ALG

Summary:

KT-Compress++ provides near-optimal compression in near-linear time

n points → Standard Thinning → $\sqrt{n}$ points with $\frac{1}{n^{1/4}}$ error (Monte Carlo rate)

n points → Kernel Thinning (KT) → $\sqrt{n}$ points with $\widetilde{O}\left(\frac{1}{\sqrt{n}}\right)$ error in n^2 time

n points → KT-Compress++ → $\sqrt{n}$ points with $\widetilde{O}\left(\frac{1}{\sqrt{n}}\right)$ error in $n \log^3 n$ time

python™ **pip install goodpoints**

arXiv.org

https://arxiv.org/abs/2105.05842 [Kernel Thinning, COLT 2021]

https://arxiv.org/abs/2110.01593 [Generalized Kernel Thinning, ICLR 2022]

https://arxiv.org/pdf/2111.07941.pdf [Distribution Compression In Near-linear Time, ICLR 2022]

Additional slides

Details of kernel thinning

KT: A two-staged algorithm

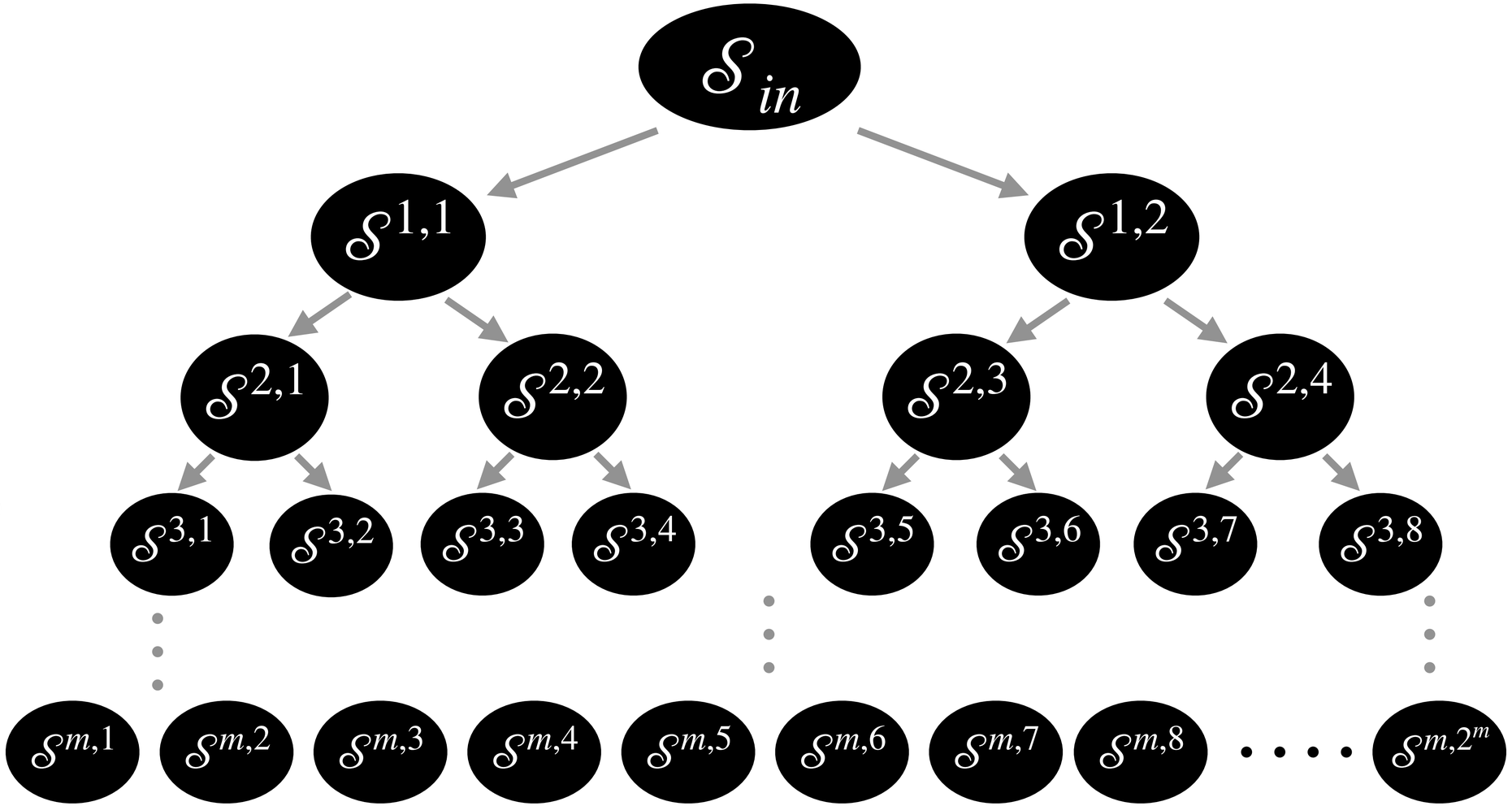
- **Input**: Kernel $\mathbf{k}$, input points $\mathcal{S}_{in}$ of size n, thinning factor m
- **KT-Split**:
  - Split $\mathcal{S}_{in}$ into 2^m balanced candidate coresets each of size $\frac{n}{2^m}$
  - When $m = \frac{1}{2}\log_2 n$, we have $\sqrt{n}$ coresets each of size $\sqrt{n}$
- **KT-Swap**:
  - Pick the best candidate coreset that minimized $\mathrm{MMD}_{\mathbf{k}}$ to input
  - Iteratively refine each point in the selected coreset by swapping with the best alternative $\mathcal{S}_{in}$ if it improves the MMD error

Computation: $\mathcal{O}(n^2)$ kernel evaluations

Storage: $n \min(n, d)$

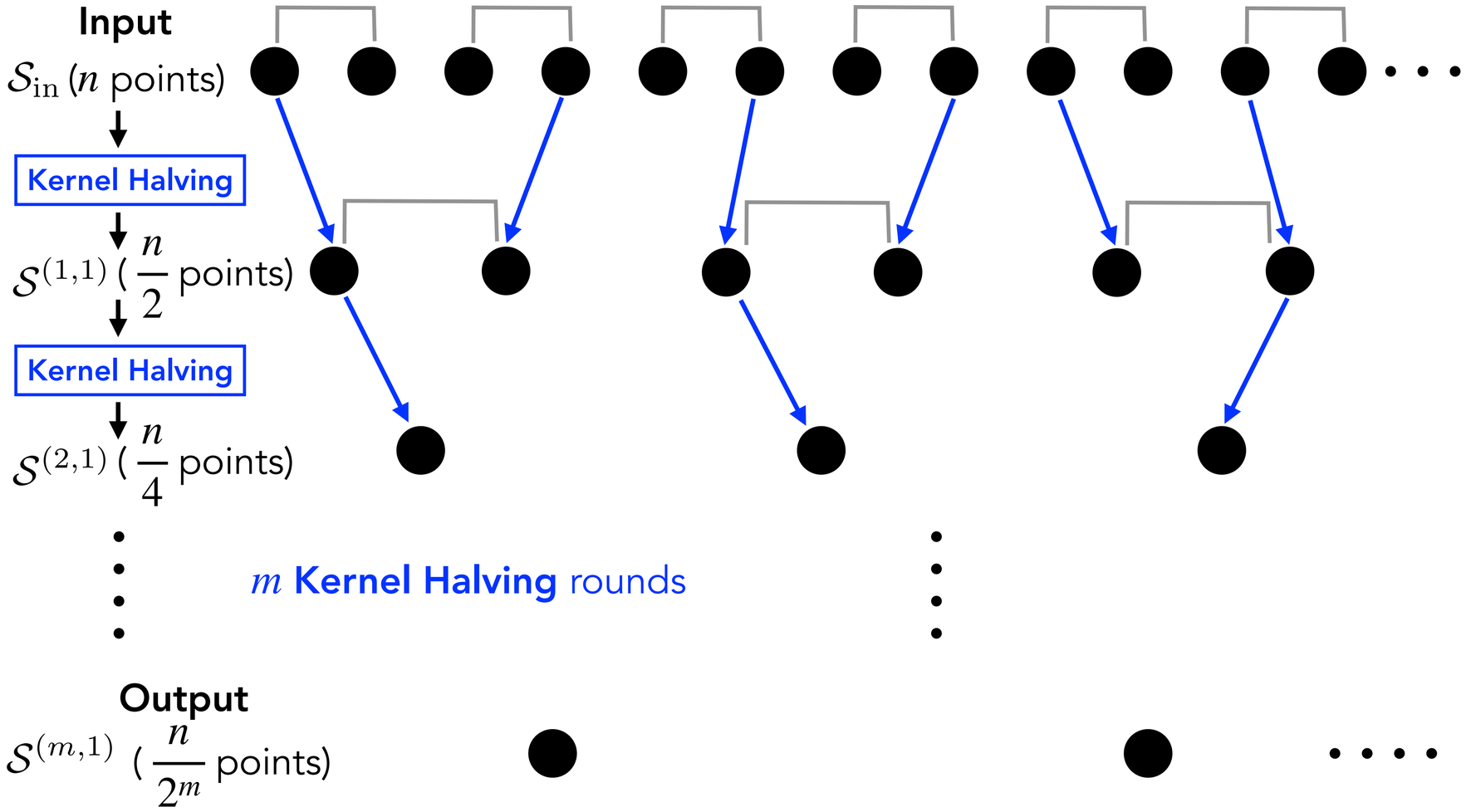
KT-Split

- Repeated rounds of splitting the parent coreset in two equal-sized children coresets
- Runs online, after seeing t input points, the bottom nodes have $t/2^m$ points



KT-Split

- One path on the tree is obtained by repeated **kernel halving**
- At each halving round, remaining points are paired, and one point is selected **non-uniformly** from each pair using a **new Hilbert space generalization of the self-balancing walk** of [Alweiss-Liu-Sawhney 2020]



Algorithm 2: Self-balancing Hilbert Walk

Input: sequence of functions $(f_i)_{i=1}^n$ in Hilbert space $\mathcal{H}$, threshold sequence $(\mathfrak{a}_i)_{i=1}^n$

$\psi_0 \leftarrow \mathbf{0} \in \mathcal{H}$

for $i = 1, 2, \ldots, n$ **do**

  $\alpha_i \leftarrow \langle \psi_{i-1}, f_i \rangle_{\mathcal{H}}$ // Compute Hilbert space inner product

  if $|\alpha_i| > \mathfrak{a}_i$**:**

    $\psi_i \leftarrow \psi_{i-1} - f_i \cdot \alpha_i / \mathfrak{a}_i$

  else:

    $\eta_i \leftarrow 1$ with probability $\frac{1}{2}(1 - \alpha_i/\mathfrak{a}_i)$ and $\eta_i \leftarrow -1$ otherwise

    $\psi_i \leftarrow \psi_{i-1} + \eta_i f_i$

end

return ψ_n, combination of signed input functions

Algorithm 2: Self-balancing Hilbert Walk

Input: sequence of functions $(f_i)_{i=1}^n$ in Hilbert space $\mathcal{H}$, threshold sequence $(\mathfrak{a}_i)_{i=1}^n$

$\psi_0 \leftarrow \mathbf{0} \in \mathcal{H}$

for $i = 1, 2, \ldots, n$ **do**

  $\alpha_i \leftarrow \langle \psi_{i-1}, f_i \rangle_{\mathcal{H}}$ // Compute Hilbert space inner product

  if $|\alpha_i| > \mathfrak{a}_i$**:**

    $\psi_i \leftarrow \psi_{i-1} - f_i \cdot \alpha_i / \mathfrak{a}_i$

  else:

    $\eta_i \leftarrow 1$ with probability $\frac{1}{2}(1 - \alpha_i/\mathfrak{a}_i)$ and $\eta_i \leftarrow -1$ otherwise

    $\psi_i \leftarrow \psi_{i-1} + \eta_i f_i$

end

return ψ_n, combination of signed input functions

We choose $\mathfrak{a}_i$ such that this step **does not occur** with high probability

Algorithm 2: Self-balancing Hilbert Walk

Input: sequence of functions $(f_i)_{i=1}^n$ in Hilbert space $\mathcal{H}$, threshold sequence $(\mathfrak{a}_i)_{i=1}^n$

$\psi_0 \leftarrow \mathbf{0} \in \mathcal{H}$

for $i = 1, 2, \ldots, n$ **do**

  $\alpha_i \leftarrow \langle \psi_{i-1}, f_i \rangle_{\mathcal{H}}$ // Compute Hilbert space inner product

  if $|\alpha_i| > \mathfrak{a}_i$**:**

    $\psi_i \leftarrow \psi_{i-1} - f_i \cdot \alpha_i / \mathfrak{a}_i$

  else:

    $\eta_i \leftarrow 1$ with probability $\frac{1}{2}(1 - \alpha_i/\mathfrak{a}_i)$ and $\eta_i \leftarrow -1$ otherwise

    $\psi_i \leftarrow \psi_{i-1} + \eta_i f_i$

end

return ψ_n, combination of signed input functions

We choose $\mathfrak{a}_i$ such that this step **does not occur** with high probability

- Exact **Kernel halving**: When $f_i = \mathbf{k}(2i, \cdot) - \mathbf{k}(x_{2i-1}, \cdot)$, exactly half of input points ($\mathcal{S}_{out}$) given -1 sign after $n/2$ steps

$$\frac{1}{n}\psi_n = \frac{1}{n}\sum_{x \in \mathcal{S}_{\text{in}}} \mathbf{k}(x, \cdot) - \frac{2}{n}\sum_{x \in \mathcal{S}_{\text{out}}} \mathbf{k}(x, \cdot) = \mathbb{P}_{\text{in}}\mathbf{k} - \mathbb{P}_{\text{out}}\mathbf{k}$$

Algorithm 2: Self-balancing Hilbert Walk

Input: sequence of functions $(f_i)_{i=1}^n$ in Hilbert space $\mathcal{H}$, threshold sequence $(\mathfrak{a}_i)_{i=1}^n$

$\psi_0 \leftarrow \mathbf{0} \in \mathcal{H}$

for $i = 1, 2, \ldots, n$ **do**

  $\alpha_i \leftarrow \langle \psi_{i-1}, f_i \rangle_{\mathcal{H}}$ // Compute Hilbert space inner product

  if $|\alpha_i| > \mathfrak{a}_i$**:**

    $\psi_i \leftarrow \psi_{i-1} - f_i \cdot \alpha_i / \mathfrak{a}_i$

  else:

    $\eta_i \leftarrow 1$ with probability $\frac{1}{2}(1 - \alpha_i/\mathfrak{a}_i)$ and $\eta_i \leftarrow -1$ otherwise

    $\psi_i \leftarrow \psi_{i-1} + \eta_i f_i$

end

return ψ_n, combination of signed input functions

- **Balance**: If $\mathbf{k}$ is a reproducing kernel, for all $g \in \mathbb{H}_{\mathbf{k}}$,

$$\langle \psi_n, g \rangle_{\mathbf{k}} = \mathbb{P}_{in} g - \mathbb{P}_{out} g \text{ is } \mathcal{O}(n^{-1} \cdot \sqrt{\log n} \cdot \|g\|_{\mathbf{k}})\text{-sub-Gaussian}$$

If η_i were chosen i.i.d., the sub-Gaussian parameter is $\Omega(n^{-1/2})$

Compress++

Compress++: A simple two-stage algorithm

n points, parameter $\mathfrak{g}$,
halving algorithm HALVE

→ Compress →

$2^g\sqrt{n}$ points

↓

$2^{\mathfrak{g}}$ thinning algorithm THIN

↓

$\sqrt{n}$ 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**
    $\widetilde{\mathcal{S}}_i \leftarrow \text{COMPRESS}(\mathcal{S}_i, \text{HALVE}, \mathfrak{g})$ // return coresets of size $2^{\mathfrak{g}} \cdot \sqrt{\frac{n}{4}}$
  end
  $\widetilde{\mathcal{S}} \leftarrow \text{CONCATENATE}(\widetilde{\mathcal{S}}_1, \widetilde{\mathcal{S}}_2, \widetilde{\mathcal{S}}_3, \widetilde{\mathcal{S}}_4)$ // 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

Compress++: A recursive strategy to reduce runtime for **generic thinning** algorithms

Thinning algorithm with n^2 runtime and error $e(n)$ → Compress++ → $\sqrt{n}$-thinning algorithm with $n \log^3 n$ runtime and error $4e(n)$

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**

    $\widetilde{\mathcal{S}}_i \leftarrow \text{COMPRESS}(\mathcal{S}_i, \text{HALVE}, \mathfrak{g})$ // return coresets of size $2^{\mathfrak{g}} \cdot \sqrt{\frac{n}{4}}$

  end

  $\widetilde{\mathcal{S}} \leftarrow \text{CONCATENATE}(\widetilde{\mathcal{S}}_1, \widetilde{\mathcal{S}}_2, \widetilde{\mathcal{S}}_3, \widetilde{\mathcal{S}}_4)$ // 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

Compress++: Informal guarantee

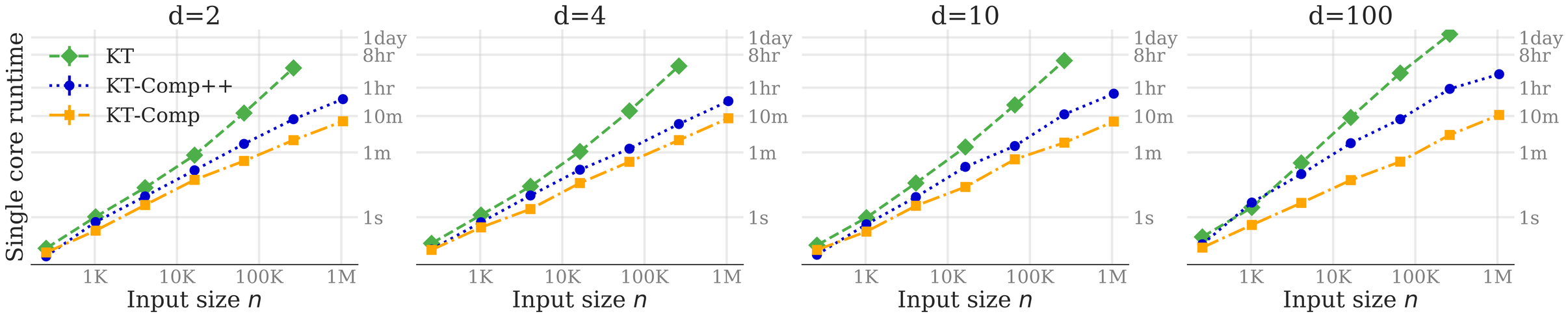
- Under some mild conditions, with $\mathfrak{g} = \log\log n + 1$, we have

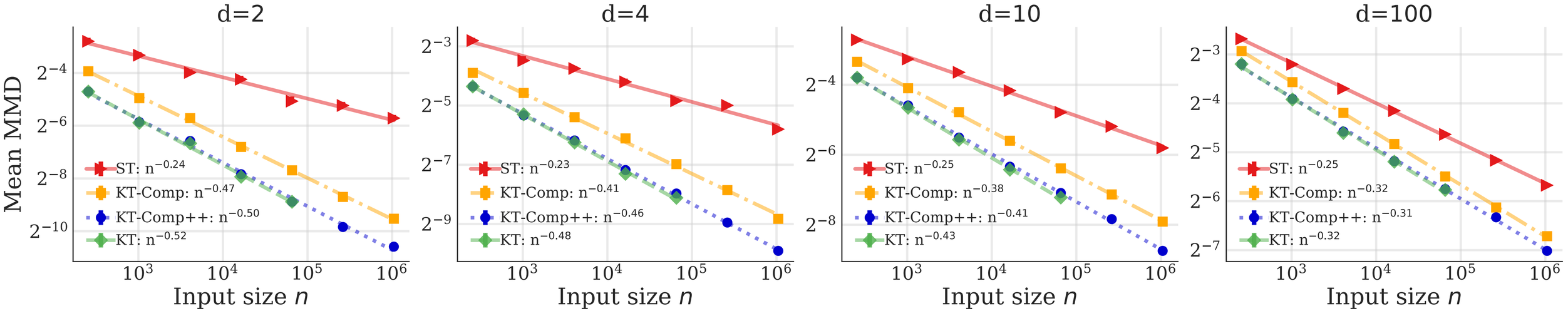
- **Error inflation by at most 4:**
  If $\text{MMD}_{\mathbf{k}}(\mathbb{P}_n, \mathbb{P}_{HALVE}) \sim e_1(n)$ and $\text{MMD}_{\mathbf{k}}(\mathbb{P}_n, \mathbb{P}_{THIN}) \sim e_2(n)$ then
  $\text{MMD}_{\mathbf{k}}(\mathbb{P}_n, \mathbb{P}_{Compress++}) \sim 4\max(e_1(n), e_2(n))$

- **Quadratic reduction in runtime**:
  If runtime of HALVE and THIN with n points is $\mathcal{O}(n^\tau)$ then the runtime of Compress++ with n points is $\mathcal{O}(n^{\tau/2})$ if $\tau > 2$ and $\mathcal{O}(n\log^3 n)$ if $\tau = 2$.

KT vs Compress ($\mathfrak{g} = 0$) vs Compress++ ($\mathfrak{g} = 4$)

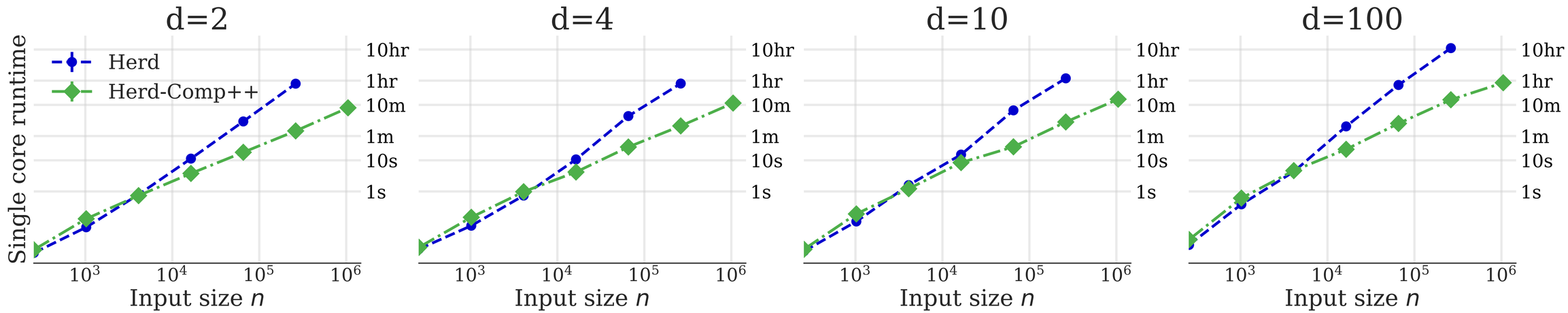
The input algorithms Halve and Thin to Compress++ are derived from KT

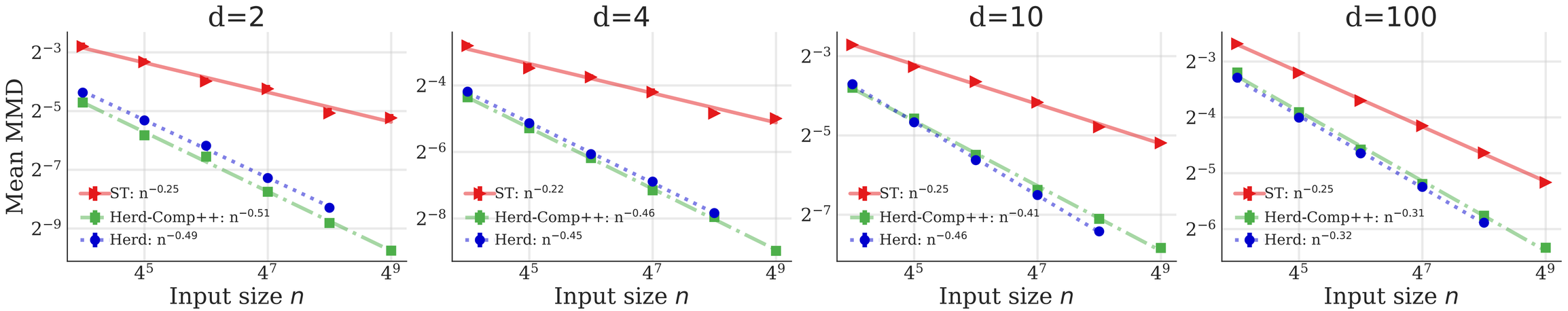




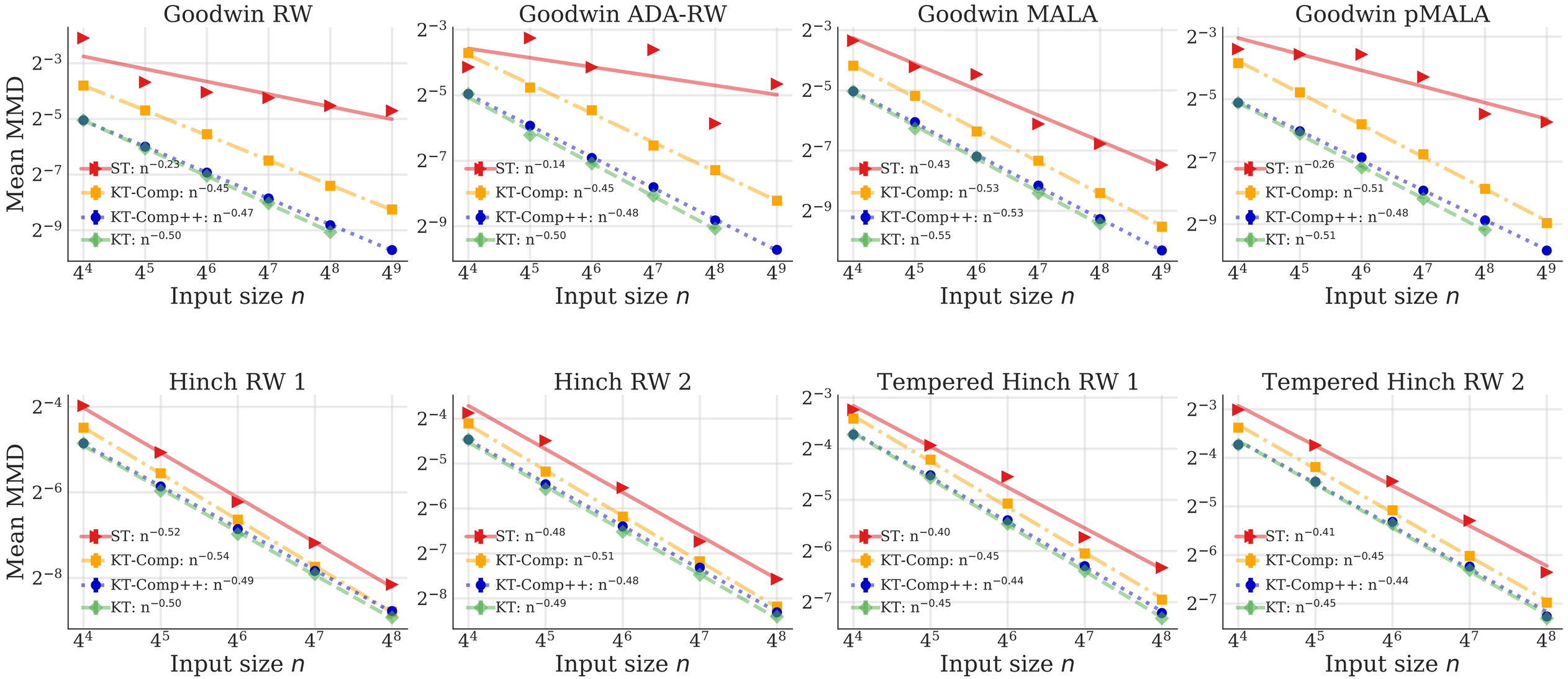
Compress++: A meta strategy to reduce runtime

Results for Compress++ with kernel herding (Herd)





MCMC Experiments



Lower bounds

Lower bounds

- For smooth kernels, there exists a target $\mathbb{P}$, such that a coreset of size $\sqrt{n}$ suffers an MMD error of $\min(\sqrt{\frac{d}{n}}, n^{-1/4})$. [Philips and Tai 2020]

- For characteristic kernels, there exists a target $\mathbb{P}$, such that any estimator based on n i.i.d. input points must suffer at least $n^{-1/2}$ MMD error. [Tolstikhin et al. 2017]

Both bounds apply to Gaussian and Matérn kernels

MCMC Details

MCMC experiments: Differential equation models

Dimension $d = 4$

1. Lotka-Volterra model
   oscillatory enzymatic control, [1925, 1926]
2. Goodwin model
   oscillatory predator-prey evolution, [1965]

MCMC experiments: Differential equation models

Dimension $d = 4$

1. Lotka-Volterra model
   oscillatory enzymatic control, [1925, 1926]
2. Goodwin model
   oscillatory predator-prey evolution, [1965]

x

1. Posterior

MCMC experiments: Differential equation models

Dimension $d = 4$

1. Lotka-Volterra model
   oscillatory enzymatic control, [1925, 1926]
2. Goodwin model
   oscillatory predator-prey evolution, [1965]

x

1. Posterior

x

1. Random walk (RW)
   [Metropolis et al. 1953, Hastings 1970]
2. Adaptive random walk (adaRW)
   [Haario et al. 1999]
3. Metropolis adjusted Langevin algorithm (MALA) [Roberts et al. 1996]
4. Preconditioned-MALA (pMALA)
   [Girolami et al. 2011]

MCMC samples taken from Riabiz-Chen-Cockayne-Swietach-Niederer-Mackey-Oates, 2021

MCMC experiments: Differential equation models

Dimension $d = 4$

1. Lotka-Volterra model
   oscillatory enzymatic control, [1925, 1926]
2. Goodwin model
   oscillatory predator-prey evolution, [1965]

Dimension $d = 38$

3. Hinch calcium signal model
   [Hinch-Greenstein-Tanskanen-Xu-Winslow, 2004]

x

1. Posterior

x

1. Random walk (RW)
   [Metropolis et al. 1953, Hastings 1970]
2. Adaptive random walk (adaRW)
   [Haario et al. 1999]
3. Metropolis adjusted Langevin algorithm (MALA) [Roberts et al. 1996]
4. Preconditioned-MALA (pMALA)
   [Girolami et al. 2011]

MCMC samples taken from Riabiz-Chen-Cockayne-Swietach-Niederer-Mackey-Oates, 2021

MCMC experiments: Differential equation models

Dimension $d = 4$

1. Lotka-Volterra model
   oscillatory enzymatic control, [1925, 1926]
2. Goodwin model
   oscillatory predator-prey evolution, [1965]

x

1. Posterior

x

1. Random walk (RW)
   [Metropolis et al. 1953, Hastings 1970]
2. Adaptive random walk (adaRW)
   [Haario et al. 1999]
3. Metropolis adjusted Langevin algorithm (MALA) [Roberts et al. 1996]
4. Preconditioned-MALA (pMALA)
   [Girolami et al. 2011]

Dimension $d = 38$

3. Hinch calcium signal model
   [Hinch-Greenstein-Tanskanen-Xu-Winslow, 2004]

x

1. Posterior
2. Tempered posterior

x

1. Random walk (RW) - run 1
2. Random walk (RW) - run 2

MCMC samples taken from Riabiz-Chen-Cockayne-Swietach-Niederer-Mackey-Oates, 2021

MCMC experiments: Differential equation models

Dimension $d = 4$

1. Lotka-Volterra model
   oscillatory enzymatic control, [1925, 1926]
2. Goodwin model
   oscillatory predator-prey evolution, [1965]

x

1. Posterior

x

1. Random walk (RW)
   [Metropolis et al. 1953, Hastings 1970]
2. Adaptive random walk (adaRW)
   [Haario et al. 1999]
3. Metropolis adjusted Langevin algorithm (MALA) [Roberts et al. 1996]
4. Preconditioned-MALA (pMALA)
   [Girolami et al. 2011]

Dimension $d = 38$

3. Hinch calcium signal model
   [Hinch-Greenstein-Tanskanen-Xu-Winslow, 2004]

x

1. Posterior
2. Tempered posterior

x

1. Random walk (RW) - run 1
2. Random walk (RW) - run 2

For KT, we use Gaussian kernel, and chose its bandwidth via median heuristic [Garreau et al. 2017]

Exact constants for KT result

Generalized kernel thinning

	Root KT	Target KT	KT+ [Best of both worlds]
KT-Split kernel	$\mathbf{k}_{rt}$	$\mathbf{k}$	$\mathbf{k} + \mathbf{k}_{\alpha-rt}$
Single-function error	Same as MMD error	$\sqrt{\frac{\log n}{n}}$ **For arbitrary $\mathbf{k}$ on arbitrary domain**	$\sqrt{\frac{\log n}{n}}$
MMD error	See slide	$\sqrt{\frac{\log^{ad+b} n}{n}}$ Analytic $\mathbf{k}$ $\sqrt{\frac{n^{d/m}}{n}}$ m-times differentiable $\mathbf{k}$	Min(Target KT Error, α-Root KT)

$\mathbf{k}(x, y) = \int \mathbf{k}_{rt}(x, z)\mathbf{k}_{rt}(z, y)dz$ & $\mathbf{k}_{\alpha-rt} = \widehat{(\widehat{\mathbf{k}})^{\alpha}}$ where $\hat{}$ denotes Fourier transform

Target KT or KT+: Better than Monte Carlo rate

$x_1, x_2, \ldots, x_n \in \mathbb{R}^d$

smooth decaying $\mathbf{k}$

$$\mathbb{P}_{in} := \frac{1}{n}\sum_{i=1}^{n} \delta_{x_i}$$

→ **Kernel Thinning (KT)** →

Non-uniform sub-sample of **size** s

$$y_1, \ldots, y_s$$

$$\mathbb{P}_{KT} := \frac{1}{s}\sum_{i=1}^{s} \delta_{y_i}$$

For any fixed $g \in \mathbb{H}_{\mathbf{k}}$, with probability $1-\delta$ over the randomness in KT, we have

$$|\mathbb{P}_{in} g - \mathbb{P}_{KT} g| \leq \frac{1}{s} \cdot \|g\|_{\mathbf{k}} \sqrt{\frac{8}{3}\|\mathbf{k}\|_{\infty} \log\left(\frac{4}{\delta}\right) \log\left(\frac{6s\log(n/s)}{\delta}\right)}$$

Much faster than the Monte Carlo rate for standard/uniform thinning $\mathcal{O}\left(\frac{1}{\sqrt{s}}\right)$

Target KT or KT+: Better than Monte Carlo rate

$x_1, x_2, \ldots, x_n \in \mathbb{R}^d$

smooth decaying $\mathbf{k}$

$$\mathbb{P}_{in} := \frac{1}{n}\sum_{i=1}^{n} \delta_{x_i}$$

→ **Kernel Thinning (KT)** →

Non-uniform sub-sample of size $\sqrt{n}$

$$y_1, \ldots, y_{\sqrt{n}}$$

$$\mathbb{P}_{KT} := \frac{1}{\sqrt{n}}\sum_{i=1}^{\sqrt{n}} \delta_{y_i}$$

For any fixed $g \in \mathbb{H}_{\mathbf{k}}$, with probability $1-\delta$ over the randomness in KT, we have

$$|\mathbb{P}_{in} g - \mathbb{P}_{KT} g| \leq \frac{1}{\sqrt{n}} \cdot \|g\|_{\mathbf{k}} \sqrt{\frac{8}{3}\|\mathbf{k}\|_\infty \log\left(\frac{4}{\delta}\right) \log\left(\frac{6\sqrt{n}\log\sqrt{n}}{\delta}\right)}$$

Much faster than the Monte Carlo rate for standard/uniform thinning $\mathcal{O}\left(\frac{1}{n^{1/4}}\right)$

Properties of MMD

- Maximum mean discrepancy (MMD) = worst-case integration discrepancy between two distributions over a class of real-valued test functions

$$\mathrm{MMD}_{\mathbf{k}}(\mathbb{P}_{in}, \mathbb{P}_{out}) = \sup_{\|g\|_{\mathbf{k}} \leq 1} |\mathbb{P}_{in} g - \mathbb{P}_{out} g|$$

[Gretton-Borgwardt-Rasch-Schölkopf-Smola, 2012]

- **MMD metrizes convergence in distribution** for popular infinite-dimensional kernels like Gaussian, Matern, IMQ, B-spline

[Simon-Gabriel-Barp-Schölkopf-Mackey, 2020]

Target KT MMD rates: $\sqrt{n}$ points with $\widetilde{O}(n^{-1/2})$ error

- More generally

$$\widetilde{O}\left(\frac{1}{\sqrt{n}}\right)$$ error rate for analytic kernels

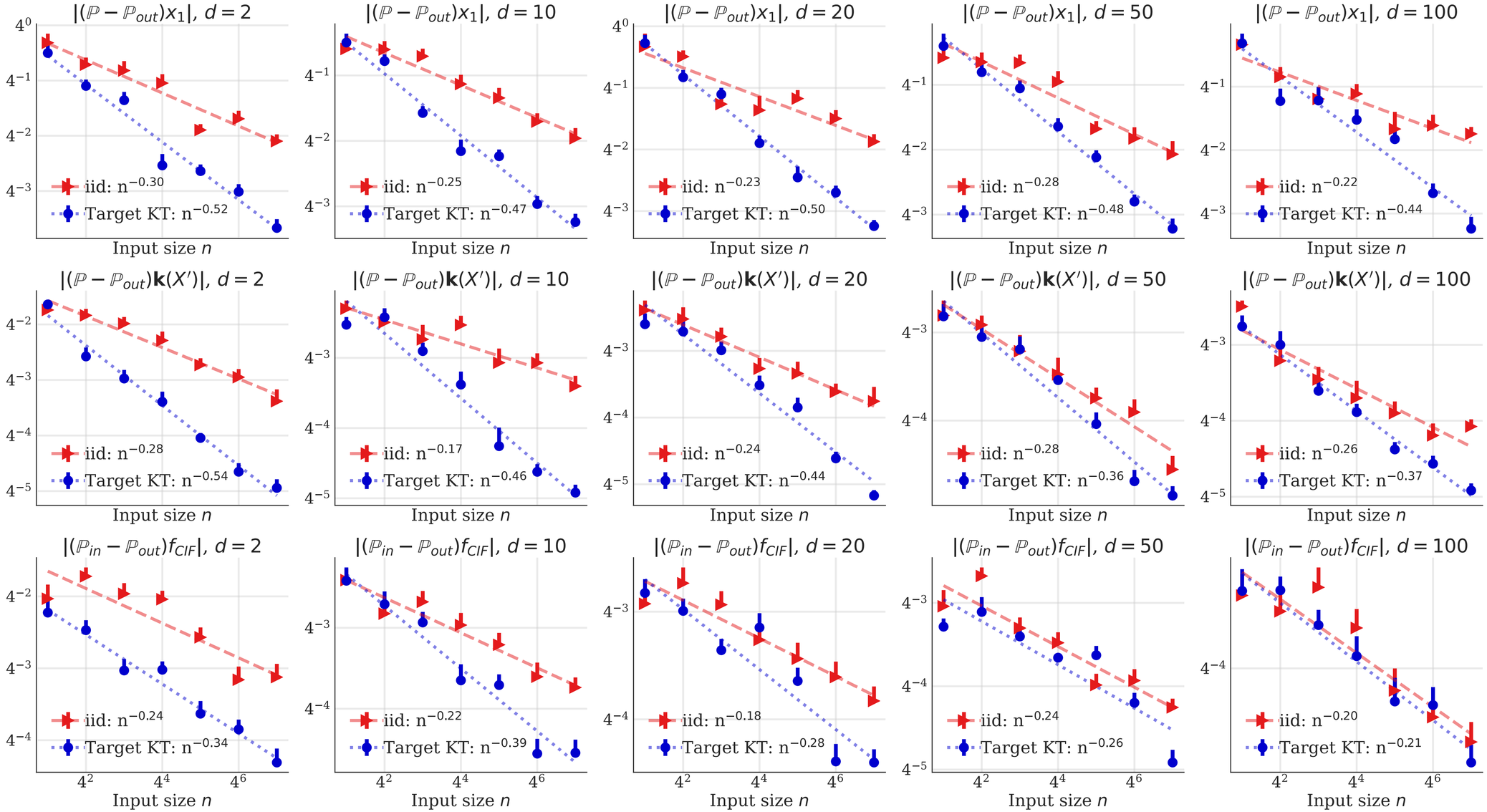
$$\widetilde{O}\left(\frac{n^{d/2m}}{\sqrt{n}}\right)$$ for m-times differentiable kernels

- We state explicit constants with dependence on kernel hyper-parameters in the paper

Single function experiments

Better error for functions inside and outside of RKHS

(Gaussian $\mathbf{k}$ with $\sigma^2 = 2d$ and standard Gaussian $\mathbb{P}^\star$)



Contrast with related work

Related work: $\sqrt{n}$ points with $n^{-1/4}$ MMD

- **Known guarantees *no better* than Monte Carlo rate**:
  Standard thinning iid points [Tolstikhin-Sriperumbudur-Muandet, 2017]
  Standard thinning geometrically ergodic MCMC [Dwivedi-Mackey 2021]
  Kernel herding for infinite-dimensional kernels [Chen-Welling-Smola 2010, Lacoste-Julien-Lindsten-Bach 2015]
  Stein Points MCMC [Chen-Barp-Briol-Gorham-Girolami-Mackey-Oates, 2019]
  Greedy sign selection [Karnin-Liberty 2019]

- ***Unknown* guarantees**:
  Support points [Mak-Joseph 2018]
  Supersampling from a reservoir [Paige-Sejdinovic-Wood, 2016]:

Related work: Better than Monte Carlo MMD

- **Finite-dimensional linear kernels:** Discrepancy construction [Harvey and Samadi, 2014]

- **Uniform $\mathbb{P}^\star$ on $[0,1]^d$ (*bounded* support):**
  Quasi Monte Carlo [Hickernell 1998, Novak-Wozniakowski 2010] ,
  Haar thinning [Dwivedi-Feldheim-Gurel-Gurevich-Ramdas 2019]

- **$\mathbb{P}^\star$ with *bounded* support with known $\mathbb{P}^\star\mathbf{k}$:**
  Bayesian quadrature [O'Hagan 1991]
  Bayes' Sard cubtature [Karvonen et al. 2018]
  Determinantal point processes [Belhadji et al. 2020]

- **($\mathbf{k}$, $\mathbb{P}^\star$) with *known/bounded* eigenfunctions:**
  Determinantal point process kernel quadrature [Belhadji et al. 2019]
  Black-box importance sampling [Liu et al. 2018]

Kernel thinning advantages

1. $\sqrt{n}$ points with $\widetilde{O}(n^{-1/2})$-MMD error (iid sampling gives $\Omega(n^{-1/4})$ error)
2. Valid for **non-uniform** target distributions with **unbounded support**
3. Valid for **infinite-dimensional** smooth/decaying kernels
4. Valid for **generic input points** including iid/MCMC/quadrature/herding with mild conditions
5. Requires **only kernel evaluations** to implement
6. **Matches MMD lower bounds** up to log factors
7. **Matches L^∞-error lower bounds** up to log factors