Rejection Sampling Optimization with Parallelization, Mixture Proposals, and Whitening

Aadit Rao
aaditrao@student.ubc.ca

Matthew Mung
mmung3@student.ubc.ca

Abstract

Rejection sampling is a simple method for sampling from difficult target distributions, but its efficiency depends strongly on the proposal distribution. We study three practical ways to improve rejection sampling without changing its core framework: vectorized and GPU-based implementations to reduce runtime, Gaussian mixture proposals for multimodal targets, and whitening for correlated targets. Using Gaussian-based test distributions, we compare these methods against a baseline sampler using runtime, acceptance rate, and sampling efficiency. Our results show that vectorization and GPU acceleration improve throughput, while mixture proposals and whitening improve proposal quality and acceptance on harder targets.

1 Introduction

Rejection sampling is a classical method for drawing samples from a target distribution when direct sampling is difficult. Its main advantages are its simplicity and its ability to produce exact samples when a valid proposal distribution and envelope bound can be found. In practice, however, its efficiency depends on how well the proposal matches the target. When the proposal is poorly aligned with the target, many candidate samples are rejected, leading to low acceptance rates. This becomes especially problematic for multimodal and highly correlated target distributions, where simple proposals may cover large regions of low-probability space.

Prior work has often improved rejection sampling by making proposals more adaptive. Gilks and Wild (1992) introduced adaptive rejection sampling for univariate log-concave densities, using piecewise exponential envelopes that tighten as sampling proceeds. Gilks, Best, and Tan (1995) extended this idea to non-log-concave settings by using a Metropolis step within the rejection procedure. More recently, Raim, Livsey, and Irimata (2024) proposed vertical weighted strips, a finite-mixture proposal for weighted densities that also provides an upper bound on rejection probability. These methods show that rejection sampling can often be improved substantially through better proposal design, but they are more specialized than the standard rejection sampler considered in this project.

In this project, we instead study simple modifications that keep the basic rejection sampling framework while improving either implementation efficiency or proposal quality. We evaluate three strategies: vectorized and GPU-accelerated implementations to reduce computational overhead, Gaussian mixture proposals to better match multimodal targets, and whitening transformations to improve sampling for correlated targets. We compare these methods against a baseline rejection sampler, using runtime, acceptance rate, and sampling efficiency as evaluation metrics. Our goal is to identify which modifications are most effective under different distributional conditions.

Course project for UBC CPSC 440/550, April 2026.

2 Methods

2.1 Baseline Setup

We evaluated our methods on target distributions with increasing levels of difficulty for rejection sampling. In one dimension, we used three targets: a standard Gaussian, a bimodal Gaussian mixture, and a more complex five-component multimodal Gaussian mixture. The Gaussian target served as a simple baseline, while the bimodal and multimodal targets were chosen to highlight situations in which a single Gaussian proposal is a poor match to the target. For the whitening experiment, we used a two-dimensional correlated Gaussian target, since correlated structure is one of the main cases where rejection sampling becomes inefficient in the original coordinate system.

For the one-dimensional experiments, we compared three proposal strategies: a baseline single Gaussian proposal, a uniform proposal, and a Gaussian mixture proposal. For each setup, we first constructed the proposal and estimated the envelope constant M over a dense grid of $10{,}000$ points. We then ran rejection sampling and recorded runtime, acceptance rate, and accepted sample count. Each one-dimensional experiment was repeated five times, and we reported median runtime and median acceptance statistics to reduce the effect of outliers and runtime noise. For visualization, we plotted the run whose acceptance rate was closest to the median so that the displayed results would be representative. In the standard one-dimensional experiments, each trial used $10{,}000$ proposals.

For the whitening experiment, we compared direct vectorized rejection sampling in the original correlated space with sampling in a whitened space and then mapped accepted samples back to the original coordinates. This experiment used $50{,}000$ proposals to make runtime and acceptance differences clearer. We also implemented a separate benchmarking pipeline for sequential, vectorized, and GPU-based sampling. These benchmark runs used larger sample counts of $10{,}000$, $50{,}000$, $100{,}000$, and $300{,}000$ proposals, again using five trials and reporting median runtime. The CuPy-based GPU benchmarks were run on a laptop with an NVIDIA GeForce RTX 2060 GPU. This setup allowed us to evaluate proposal quality and implementation efficiency separately.

2.2 Vectorized and GPU-Accelerated Sampling

Our first optimization focused on improving the runtime of rejection sampling through vectorization and GPU acceleration. In the baseline implementation, candidate samples were generated and tested one at a time in a Python loop. For each proposal, the sampler drew a candidate x, generated a uniform random value u, computed the acceptance probability $\frac{f(x)}{Mg(x)}$, and then accepted or rejected the sample. Although this procedure is simple, it introduces substantial Python overhead when many proposals are required.

To reduce this cost, we implemented a vectorized NumPy version that generates a batch of proposals and random values at once, computes acceptance probabilities as arrays, and uses a boolean mask to keep accepted samples. We extended the same approach to the GPU using CuPy, replacing NumPy operations with GPU-compatible ones. This also required corresponding CuPy versions of the target density functions, since NumPy-based PDFs cannot operate directly on CuPy arrays.

This optimization was applied to the Gaussian, uniform, and mixture proposal samplers. Its purpose was not to change the statistical behavior of rejection sampling, but to improve computational throughput by reducing loop overhead and exploiting parallel array operations. Therefore, we expected vectorization and GPU acceleration to leave acceptance rates essentially unchanged while significantly reducing runtime, especially for larger batch sizes.

2.3 Gaussian Mixture Proposals

Our second optimization focused on improving proposal quality by replacing a single Gaussian proposal with a Gaussian mixture proposal. A single Gaussian can work well for simple unimodal targets, but it is often a poor fit for bimodal or multimodal distributions. In those cases, the proposal must be scaled by a larger constant M to cover all regions of the target, including the low-density space between modes. This lowers the acceptance rate and causes many samples to be wasted in regions where the target density is small.

To address this, we constructed the proposal as a weighted mixture of Gaussians whose components were estimated directly from the target density over a fixed range. We first evaluated the target on a dense grid and used SciPy's peak detection routine, `scipy.signal.find_peaks`, to identify its main modes. The detected peak locations were used as the mixture means, while the peak heights were normalized to define the component weights. The component standard deviations were then estimated from the spacing between peaks and scaled slightly to provide enough overlap and coverage across the full target. If no distinct peak was found, the proposal defaulted to a single component centered at the maximum of the target density.

Once the mixture proposal was defined, the rest of the rejection sampling algorithm was the same. A component was first selected according to its mixture weight, then a sample was drawn from that Gaussian component, and finally the usual rejection rule was applied using the ratio between the target density and the mixture proposal density. We also computed the envelope constant M numerically on a dense grid to ensure that the proposal properly bounded the target over the region of interest.

Unlike the vectorization and GPU optimization, this method was meant to improve acceptance rate rather than raw runtime. Its main benefit is that it shapes the proposal to better match the geometry of multimodal targets, reducing wasted sampling between modes and increasing overall efficiency. As a result, we expected this optimization to be most helpful on the bimodal and complex multimodal targets, where the limitations of a single Gaussian proposal are most apparent.

2.4 Whitening for Correlated Targets

Our third optimization focused on improving rejection sampling for correlated targets through whitening. Even when the target is unimodal, rejection sampling can still be inefficient if the distribution is stretched or strongly correlated across dimensions. In the original coordinate system, a simple isotropic Gaussian proposal may cover a large amount of low-probability space, which reduces acceptance rate and wastes samples.

To address this, we transformed the target into a whitened coordinate system where the correlation structure is removed. Using the Cholesky factor L of the covariance matrix, we mapped samples from the original space x into whitened coordinates y, where the correlated Gaussian target becomes approximately isotropic. In this transformed space, it is much easier to use a simple Gaussian proposal that aligns well with the target. After performing rejection sampling in the whitened space, the accepted samples were mapped back to the original coordinates for visualization and comparison.

In our implementation, we compared this whitened sampler against a baseline two-dimensional Gaussian proposal applied directly in the original correlated space. Both versions used vectorized rejection sampling, but the whitening transformation changed the geometry of the problem so that the proposal could match the target more effectively. As with the mixture proposal, this optimization was intended to improve proposal quality rather than implementation speed. Its main benefit is that it reduces inefficiency caused by correlation and anisotropy in the target distribution.

3 Experiments and Analysis

3.1 Runtime Benchmarks

The runtime benchmarks show that vectorization and GPU acceleration both greatly reduce the cost of rejection sampling, while leaving acceptance rates essentially unchanged. At a smaller batch size of $n = 10{,}000$ (Figure 1), the vectorized NumPy implementation was the fastest across all benchmark cases. Compared with the sequential implementation, vectorization gave speedups ranging from about $65.9\times$ to $356.5\times$, while the GPU version still improved over sequential runtime but remained slower than NumPy vectorization. This is most likely due to GPU launch and memory overhead dominating at smaller problem sizes. For example, on the baseline Gaussian target, sequential sampling took $0.0383\,\mathrm{s}$, the vectorized version took 5.80×10^{-4} s, and the GPU version took 1.20×10^{-3} s. A similar trend appeared across the other proposal types.

At the larger batch size of $n = 300{,}000$ (Figure 2), the results changed substantially. In this case, the GPU implementation became the fastest in every benchmark case, showing that the overhead of moving work to the GPU is outweighed once the batch size is sufficiently large. GPU speedups

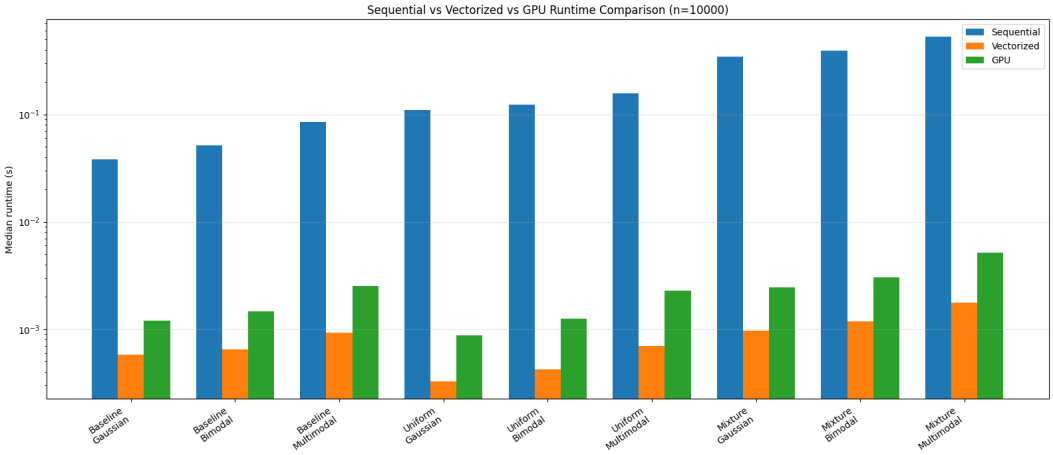


Figure 1: Runtime Benchmark n=10000

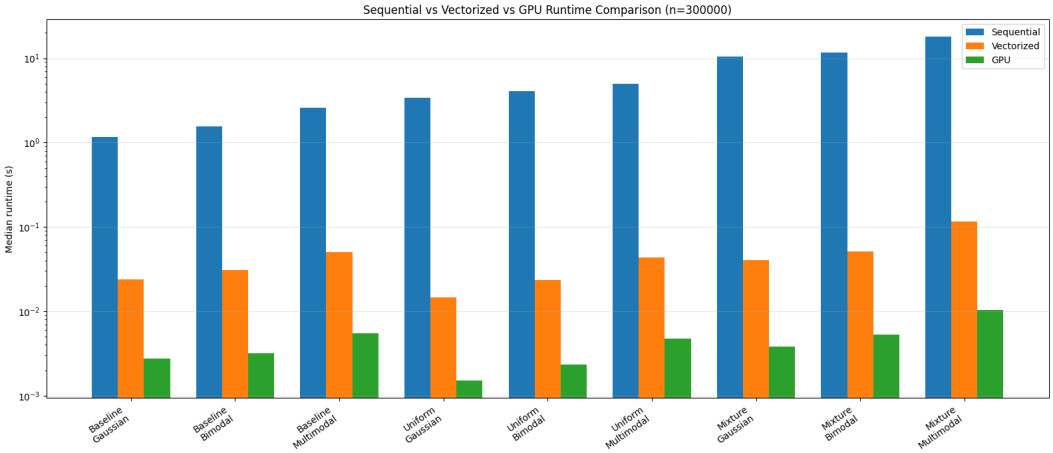


Figure 2: Runtime Benchmark n=300000

over the sequential implementation ranged from about $420\times$ to $2727\times$, while the vectorized NumPy version achieved speedups between about $48\times$ and $259\times$. This effect was especially strong for the more expensive uniform and mixture samplers. For instance, on the mixture multimodal benchmark, runtime dropped from 18.01 s in the sequential case to 0.116 s with NumPy vectorization and 0.0104 s on the GPU, corresponding to a GPU speedup of about $1731\times$. On the uniform Gaussian benchmark, runtime dropped from 3.43 s to 0.0146 s and then to 0.00150 s, yielding a GPU speedup of about $2278\times$.

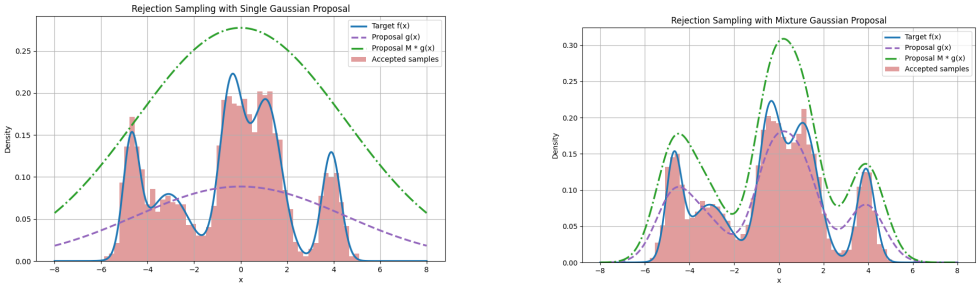
Across all benchmark sizes, acceptance rates were nearly identical for the sequential, vectorized, and GPU implementations. For example, at $n = 300{,}000$ on the baseline Gaussian target, the acceptance rates were 0.6351, 0.6348, and 0.6349 respectively. This consistency confirms that the optimization changes only the implementation efficiency of the sampler rather than the underlying sampling behavior. Overall, these results show that NumPy vectorization already provides large runtime gains at modest batch sizes, while GPU acceleration becomes most beneficial once enough proposals are processed to amortize device overhead.

3.2 Mixture Proposal Results

The mixture Gaussian proposal improved acceptance rates substantially, especially on targets with multiple modes. While a single Gaussian proposal can work reasonably well for simple unimodal targets, it becomes increasingly inefficient as the target develops separated peaks and irregular structure. This trend is reflected clearly in our results. On a single Gaussian target, the acceptance

rate improved from 0.6338 with the single Gaussian proposal to 0.7347 with the mixture proposal. On the bimodal target, it improved from 0.4139 to 0.5370. The largest improvement occurred on the complex multimodal target, where the acceptance rate increased from 0.3180 to 0.5844. For reference, the uniform proposal achieved only 0.2695 on this same target.

The multimodal case best illustrates why this improvement occurs. With a single Gaussian proposal (Figure 3a), the sampler must use a large envelope constant, $M = 3.1309$, in order to cover all peaks of the target distribution. As a result, many samples are drawn in low-density regions between modes and end up being rejected. In contrast, the mixture proposal (Figure 3b) achieved a much smaller envelope constant of $M = 1.7050$, showing that it matched the target density much more closely. The detected component means matched the visible peaks of the target well, so the proposal placed its probability mass in the right regions.



(a) Single Gaussian proposal

(b) Mixture Gaussian proposal

Figure 3: Rejection sampling on the complex multimodal target using a single Gaussian proposal and a Gaussian mixture proposal.

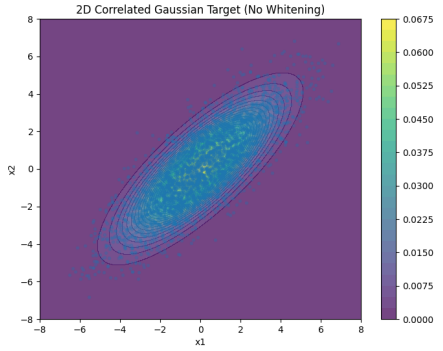
This difference is also visible in the sampling plots. In the single Gaussian case, the proposal is broad and does not reflect the target's multimodal structure, so the envelope sits far above the target over much of the domain. In the mixture case, the proposal follows the individual modes more closely, and the histogram of accepted samples matches the target much better with less wasted space between peaks. Although the mixture sampler was more expensive per proposal than the simpler samplers, its main purpose was to improve the proposal itself, and the higher acceptance rates on the bimodal and multimodal targets show that it did.

3.3 Whitening Results

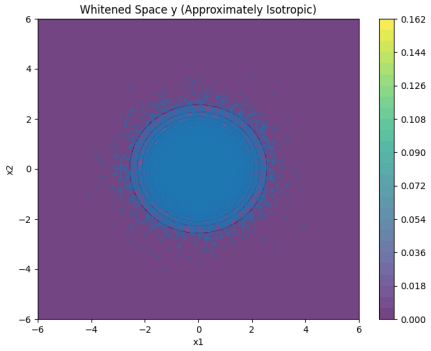
The whitening transformation improved rejection sampling performance substantially on the correlated two-dimensional Gaussian target. Without whitening, the baseline sampler in the original x-space required a large envelope constant of $M = 10.5594$, accepted only 4650 samples out of $50{,}000$, and achieved an acceptance rate of 0.0930. After transforming the target into the whitened y-space, the envelope constant dropped to $M = 2.3613$, the number of accepted samples increased to 21068, and the acceptance rate rose to 0.4214. Runtime also improved slightly, decreasing from 0.0152 s to 0.0106 s.

These results show that the main benefit of whitening was not simply computational speed, but improved proposal quality. In the original coordinate system, the target is stretched and highly correlated, so a simple isotropic Gaussian proposal covers a large amount of low-density space and leads to many rejections. After whitening, the target becomes much closer to isotropic, allowing the same style of Gaussian proposal to match the target much more closely. This is reflected in the much smaller value of M and the large increase in acceptance rate.

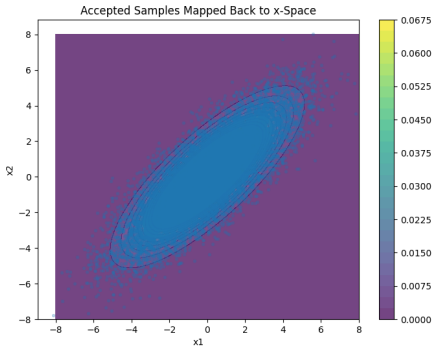
The geometric effect of whitening is also clear in the plots. In the original x-space (Figure 4a), the target has a long tilted elliptical shape, which makes it difficult for an isotropic proposal to cover efficiently. In the whitened space (Figure 4b), the target becomes approximately circular, making rejection sampling much more efficient. After the accepted samples are mapped back to the original coordinates (Figure 4c), they again match the correlated elliptical structure of the original target. This confirms that whitening improves efficiency during sampling while still producing samples in the correct original distribution.



(a) Direct rejection sampling in the original x-space without whitening.



(b) The same target represented in the whitened y-space.



(c) Accepted samples from the whitened sampler mapped back to the original x-space.

Figure 4: Effect of whitening on rejection sampling for a correlated two-dimensional Gaussian target

4 Discussion and Future Work

4.1 Main Conclusions

In this project, we studied several practical ways to improve the efficiency of rejection sampling without changing its underlying framework. Starting from a baseline rejection sampler, we evaluated three optimizations: vectorized and GPU-accelerated sampling, Gaussian mixture proposals, and whitening for correlated targets. Each method addressed a different weakness of standard rejection sampling, allowing us to study both implementation efficiency and proposal quality.

Our results showed that vectorization and GPU acceleration greatly reduced runtime while leaving acceptance rates essentially unchanged. The mixture Gaussian proposal improved acceptance rates substantially on bimodal and multimodal targets by matching the target structure more closely than a single Gaussian proposal. Whitening also proved effective for correlated two-dimensional targets, where transforming the distribution into a less correlated space led to a much smaller envelope constant and a large increase in acceptance rate.

Overall, these results show that rejection sampling can be improved significantly through relatively simple but well-motivated modifications. Rather than proposing a new sampling method, our work demonstrates that better proposal design and more efficient implementations can make classical rejection sampling much more practical in difficult settings.

4.2 Strengths and Limitations

A strength of this project is that it compares several distinct types of improvements within a common experimental framework. Instead of focusing only on faster implementation or better proposal design, we studied both and showed how they affect different aspects of rejection sampling performance. This made it possible to separate improvements in computational throughput from improvements in acceptance rates.

One limitation of this project is that our experiments were carried out on a relatively small set of target distributions. While these targets were chosen to highlight important weaknesses of standard rejection sampling, they were still mostly low-dimensional and analytically simple. In particular, our whitening experiment focused only on a two-dimensional correlated Gaussian, and our mixture proposal experiments were limited to one-dimensional multimodal targets.

Another limitation is that the performance study was conducted on a limited range of hardware and benchmark settings. Our GPU experiments were run on a computer with an NVIDIA GeForce RTX 2060 GPU, so the runtime results may differ on stronger desktop GPUs or on systems with different memory and transfer overheads. We also used five trials per setting and reported median runtime and acceptance statistics, which gave stable comparisons, but more trials would allow for more thorough results on runtime variability.

A further limitation is that our proposal construction methods were still fairly simple. For the mixture proposal, the Gaussian components were estimated using peak detection and simple heuristics for the component widths. For whitening, we assumed that the covariance structure of the target was known or could be specified directly. These choices were reasonable, but they limit how broadly the results can be generalized to more realistic sampling problems.

4.3 Future Directions

With more time, we would test these methods on a wider range of target distributions, including higher-dimensional correlated targets, more irregular multimodal distributions, and targets whose structure is less cleanly captured by Gaussian components. This would help determine how well the observed improvements behave beyond the relatively simple distributions considered here.

We would also expand the benchmarking across different CPUs and GPUs, test larger sample sizes, and measure additional performance metrics such as accepted samples per second and the cost of host-device data transfer separately. Together, these extensions would provide a clearer picture of when the proposed optimizations are most beneficial in practice.

References

W. R. Gilks, N. G. Best, and K. K. C. Tan (1995). "Adaptive Rejection Metropolis Sampling Within Gibbs Sampling." *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 44.4, pp. 455–472. DOI: 10.2307/2986138.

W. R. Gilks and P. Wild (1992). "Adaptive Rejection Sampling for Gibbs Sampling." *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 41.2, pp. 337–348. DOI: 10.2307/2347565.

Andrew M. Raim, James A. Livsey, and Kyle M. Irimata (2024). "Rejection Sampling with Vertical Weighted Strips." DOI: 10.48550/arXiv.2401.09696. arXiv: 2401.09696.