Parallelizing Bayesian Trees: Algorithmic Insights from GPU BART

Howell Lu [1]

Abstract

In the past decades, many algorithms have been modified to run on Graphics Processing Units (GPUs) to utilize increased processing power. Neural networks and convolutional networks are some prominent examples. These changes have occurred due to the development of new platforms, such as Compute Unified Device Architecture (CUDA), which supports optimized libraries for low-level matrix computations like cuBLAS, which enables the development of high-level frameworks such as JAX. This has spurred the development of branchless and parallelizable versions of preexisting algorithms. BART (Bayesian Additive Regression Trees) is a sum of tree methods released in 2010 which has been frequently used for non-parametric estimation due to inference proficiency. The literature around BART algorithm has expanded greatly within the prior decade with many implementations focused on accelerated computing. BARTZ, also referred to as GPU-BART (Super-Fast Bayesian Additive Regression Trees) is one recent addition, which utilizes GPU computation, enabling speed-ups of an order of magnitude on large datasets. We attempt to both analyze and rigorously stress-test BARTZ to determine whether modifications are extendable to the domain of causal inference. After rigorous testing, BARTZ matches the performance of commonly used libraries for nearly all surfaces, only struggling with a few theoretical surfaces and edge cases.

1. Introduction

Bayesian Additive Regression Trees (BART) (Chipman et al., 2010) is a close relative of the decision tree family, which utilizes branching decision rules to split data into groups, generally to predict against a held off dataset. The progenitor of the BART family was Classification and Regression Trees (CART) (Breiman et al., 1984) which introduced predictive splitting on trees. For instance, a split might assign a predicted height of 175cm to individuals over the age of 20 and 160cm to those under the age of 20. Although flexible, this was prone to overfitting. Further libraries such as Random Forest (Breiman, 2001) and XGBoost (Chen & Guestrin, 2016) were designed to solve this overfitting. Random Forest addressed overfitting through the use of an ensemble (average prediction) of multiple trees, each trained on a bootstrap sample of the data. XGBoost used a sum of trees model wherein the first tree predicts the model and each subsequent tree predicted on the residuals of the preceding tree with the final model as the sum of these predictions. BART shares structural aspects from both of its relatives; it utilizes multiple trees like Random Forest, but is sequential like XGBoost as an inter-tree dependency exists. However, BART is uniquely Bayesian and thereby more difficult to parallelize as it probabilistic samples utilizing an MCMC format with each tree being conditional of the residuals of other preceding trees.

XGBoost and Random Forest have been translated into GPU-compatible versions (Mitchell & Frank, 2017) (Liao et al., 2013) with both showing nearly 100x speed-ups on large datasets as compared to the traditional CPU versions. These algorithms have managed to replicate the outputs of their CPU counterparts with negligible loss in accuracy. These are the specific changes done:

1. Utilization of Binning Instead of Direct Splits (XGBoost and Random Forest): GPUs are extremely poor at sorting between features and moving observations into nodes. However, GPUs are exceptionally fast at computing. If data is categorized into ordered bins beforehand, rather having to sort through all observations and to assign all observations into specific leaf nodes, one can simply map bins to nodes rather than individual observations. This can lead to problems if the distribution of the independent variables are extremely tail ended and if relationships between the independent variables and the outcome variables are heterogenous. However, it should be noted that although utilizing every split within a GPU is virtually impossible, CPU tree algorithms generally prefer to also use binning to

[*]Equal contribution [1]Steinhardt, New York University, New York, United States. Correspondence to: Howell Lu <hl4631@nyu.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

minimize compute time, despite having the ability to use all splits.

2. Further Partitioning by Bins and Nodes by Threads : The GPU versions of other Tree methods allow threads to split the data into partitions to be computed individually. If one were to have a outcome variable, a single predictor and a single tree, one thread would compute the residuals for half the bins and another thread the other half. This level of granularity applies within nodes as an individual thread can calculate the residuals for observations within a node. It should be noted that standard CPU Random Forest does not partition between bins, standard XGBoost does. However, only the GPU versions of both have computing on the thread-observation level.

3. Switching the dataset from a linked tree structure to a continuous memory layout (Array): Both algorithms declare memory statically, before the initiation of the algorithm. Removing the need to constantly need to deal with the remapping of observations between nodes.

Several BART implementations have attempted to utilize some of the prior changes for accelerated computing. The first generation was PBART (Pratola et al., 2014)(Parallel BART), which implemented the partitioning of trees, bins and observations (did not change the memory format) into different threads, allowing specific threads to compute on specific CPU cores. These output of these threads would be compiled by a "Master Core" after computing. The limitation of this format was that such trees were trained on dated trees from the last MCMC sample rather than within an MCMC cycle, thereby omitting the sequential dependence factor.

One later iteration is XBART (He & Hahn, 2023), which utilized a stochastic hill-climb in lieu of MCMC sampling. This process limited the algorithm to use bins, allowed the partitioning of trees, bins and nodes by Threads and switched the format from a linked tree to an array. However, such process removes the MCMC chain and therefore loses its bayesian properties, in exchange for removing sequential dependence.

The complexity of paralleling BART comes from the paradoxical problem of needing to parallelize a sequential and probabilistic algorithm. Both previous iterations traded sequentialism and probabilism in favor of computability. However, BARTZ succeeds in maintaining sequentialism and probabilism while making these changes, by precomputing and redundant calculation. However, these modifications mentioned above are not exhaustive and the initial GPU-BART implementation has not been exhaustively tested for especially on difficult response surfaces and for the causal inference usage.

2. The Core Aspects of BART and it's limitations:

BART or (Bayesian Additive Regression Trees) is a sum of trees model which updates a fixed set of trees repeatedly. BART differs from other tree methods as it samples stochastically rather than greedily.

2.1. A high-level overview of the BART algorithm:

1. Define parameters and priors such as the number of trees, the total amount of samples, the priors on the tree structure, the prior on the leaf values and the priors on the residual variance.

2. Select a proposal (GROW/PRUNE/CHANGE/SWAP) for a given tree.

3. Calculate the posterior odds for this proposal:
   (a) Identify which node each observation belongs to
   (b) Calculate the residuals for all trees other than this tree (remove the current tree predictions).
   (c) Utilize the residuals to calculate the marginal likelihood.
   (d) Multiply the likelihood by the prior and a transition adjustment.

4. Sample the posterior odds, depending on the odds, we either probabilistically accept or reject the transition.

5. Traversing, deleting the redundant tree pointers and linking new tree pointers (if accepted)

6. Sample the leaf parameters using the new tree distribution and the priors for predictions.

7. Recalculate the residuals to have a running count of the current tree.

8. Move to the subsequent tree and repeat steps 2-7 for the subsequent tree within this sample.

9. After completing a full MCMC sample, sample a new global residual variance from the tree ensemble from the most recent ensemble.

10. Complete steps 2 to 9 for the remaining number of selected MCMC runs.

11. Utilize this model for predictions.

Each tree update requires the updated residuals of the previous tree, as Bayesian Backfitting is circular or double nested. Each tree must be built up in sequential order, and each sample must also run in order. Thereby, the process of generating a BART tree requires sequential processing of $N_{\text{trees}} \times N_{\text{samples}}$, making it far less parallelizable than

its tree neighbors. Skipping this sequentiality makes BART lose the Bayesian aspect.

Step 2 is probabilistic as one does not know whether the proposition will be "GROW" or "PRUNE". Having different threads perform a different operation creates a significant slow-down.

2.2. The Computational Bottlenecks of Traditional BART

If one were to run a hypothetically large dataset, here are some computational scaling estimates:

Table 1. Estimated Compute Time and Scaling

Step Description	Scaling
Steps 3a, 3b, & 7	N
3a. Mapping observations to nodes 3b. MCMC Residual calculation 7. Recalculating ensemble tree	*(95% of compute)*
Step 5	$\log(N)$
5. Traversing, deleting the redundant tree pointers and linking new tree pointers (if accepted)	
Everything Else	$O(1)$

The residuals and the observation-to-tree mapping are the most computationally difficult aspect of BART due to the scale of the computation; each observation needs to be relinked to a tree node and each observation must have its residuals calculated repeatedly. All other calculations are far less frequent; such as the likelihood calculation, the tree proposal choices which only occurs once per tree-sample iteration.

A second computational bottleneck is the need to modify its tree structure. Unlike other tree algorithms, where the nodes and pointers of a single tree are generally fixed after creation; BART constantly requires the creation, modification and relinkage of memory pointers within every MCMC proposal. The operating system must delete and assign memory, a computationally slow task, for every tree proposal.

3. Unique Characteristics of BARTZ

BARTZ succeeds in achieving substantial speedups by utilizing a series of modifications which increase the speed of the algorithm.

1. Tree modifications are limited to grow and prune, and both pathways are computed, with only one being selected at random. This solves the warp divergence issue.
2. Proposal computation and mapping are offloaded to separate threads. The proposal moves and the observation-node mapping do not require sequential data from preceding trees, but only the last iteration of the present tree. The GPU can build given trees and map observations to nodes in parallel on different threads. The residual calculations and acceptance stages are later calculated sequentially after all the parallel trees are built. This process allows BARTZ to parallelize the majority of the sequentialness of BART.
3. The data representation changes from being a binary tree to being a three arrays: the axis, cutpoint and value.
4. The computations are parallel across data (where applicable): Each Tree, bin, and observation can be computed by different threads, in a similar manner to GPU-Random Forest.
5. A reduction in precision from 64-bit to 32-bit values.
6. Forcing all potential splits within independent variables to be a pre-defined bin rather than sorting and utilizing every potential split.

4. BARTZ Limitations

The prior changes enable BARTZ to run efficiently, branchless and in parallel. However, there are limitations to these modifications. Many of these modifications are already a preferred practice with traditional BART, such as binning (numcut). Other changes can create potential issues.

4.1. Exponential Growth of Memory and the Depth Limitation

BARTZ declares all memory upfront and modifies preallocated arrays for computational needs. This is extremely computationally efficient, as pointers do not need to be changed. Furthermore, traditional BART leaf nodes require 64 bytes of memory to create a leaf node but BARTZ can use a fraction of that memory in specific circumstances, such as low depth trees, by creating a representation of a leaf node with only 4 bytes.

However, BARTZ's lower-bound and upper-bound memory usage is identical as memory is declared statically rather than dynamically. Thereby, the same amount of memory is utilized regardless of whether such memory is used, which contrasts to traditional BART libraries which only use memory when creating trees which are used.

An extreme example would be a tree with a depth of 20, wherein all left nodes are leaf nodes and all right nodes continue to branch. The lower bound of a classical BART tree

representation of this, would be 64×21 bytes $= 1{,}344$ bytes .However, that exact same structure within a BARTZ framework would be $2^{20} \times 16$ bytes ≈ 16 MB (more expensive data types are needed with increasing depth) as the entire memory block must be statically declared, taking over 10,000 times the memory.

BARTZ limiting tree depth to prevent this issue, which raises a critical question: Do these constraints restrict the modeling of complex, and high-interaction response surfaces?

4.2. Limiting the Tree Choices to Grow Prune

BART with GROW-PRUNE alone, is known to be ergodic, meaning that all possible trees will eventually be reached and an optimal tree will be devised. However, the discovery time of a solution for a response surface is highly dependent on the sampler's options and it is entirely possible for some response surfaces to be extremely difficult to solve or stuck within a local minimum with a limited grow/prune selection compared to a 4-stage sampler (GROW/PRUNE/CHANGE/SWAP).

4.3. Limiting the Precision to 32-bit

Modern GPU's have a different architecture than CPUs. Each GPU core is designed to operate at a certain level of precision which cannot be changed. Modern GPUs are built with a majority of 32-bit cores, causing 64-bit computation to run at a fraction of the speed of 32-bit. This may present issues with overflow and underflow inaccuracies for computations for the likelihood and the posterior.

4.4. Limitations to the Minimal Tree Size

Both BART (2021) (Sparapani et al., 2021) and BARTZ implemented of a limitation on node-size. Both frameworks only accept splits on nodes with 10 or more observations, thereby reducing all leaf nodes to a minimum of 5 observations. This prevents overfitting for outliers, but makes the capture of simple response surfaces difficult. Additionally, small datasets may be disproportionally affected when the minimal leaf node represents a large percentage of the total observation population, leading to inaccuracies. Furthermore, outliers which have irregular, rugged effects cannot be properly individually modeled.

This is a simple response surface demonstrates this problem by causing BARTZ and BART (2021) to underperform:

Let us represent a dataset with predictors like:

$$X_{i,j} \sim \mathcal{U}\{20, 80\} \quad \text{for } i = 1, \ldots, 5004,\ j = 1, \ldots, 3$$

$$X_{i,4} = \begin{cases} \sim \mathcal{U}\{20, 80\} & i \leq 5000 \\ \sim \mathcal{U}\{-1000, -500\} & i \in \{5001, 5002, 5003, 5004\} \end{cases}$$

$$X_{i,5} = \begin{cases} \text{Bern}(0.5) & i \leq 5000 \\ 1 & i \in \{5001, 5002\} \\ 0 & i = \{5003, 5004\} \end{cases}$$

$$Y_i = (X_{i,1} + X_{i,2} + X_{i,3} + X_{i,4}) + 4 \cdot X_{i,5}$$

This creates a simple additive response surface wherein, there exists a few outliers for X_4, and X_5 is the treatment variable creating an ITE for 4 for each individual.

We also devised a smaller version of this dataset with the same calculation for the outcome variable but with substantially fewer observations

$$X_{i,j} \sim \mathcal{U}\{20, 80\} \quad \text{for } i = 1, \ldots, 104,\ j = 1, \ldots, 3$$

$$X_{i,4} = \begin{cases} \sim \mathcal{U}\{20, 80\} & i \leq 100 \\ \sim \mathcal{U}\{-1000, -500\} & i \in \{101, 102, 103, 104\} \end{cases}$$

$$X_{i,5} = \begin{cases} \text{Bern}(0.5) & i \leq 100 \\ 1 & i \in \{101, 102\} \\ 0 & i = \{103, 104\} \end{cases}$$

We compare the results of BARTZ with a modified version of the BARTZ library with node restrictions removed and DBarts (Dorie et al., 2025) which has no leaf node restriction size. We utilized default BART parameters for all samplers, and give BARTZ a maximum depth of 8.

Table 2. Comparison of GPU BART and DBarts Performance (Full Dataset)

	GPU BART	**GPU BART (Node Restrictions Removed)**	**DBarts**
RMSE	3.942	1.296	0.903
Estimate (Outl.)	-23.81	1.370	9.193
Estimate	4.013	4.010	4.016
Bias (Outl.)	-27.812	-2.63	5.193
Bias	0.013	0.010	0.016

The outliers cannot separated individually, and thus BARTZ cannot provide reasonable ITE estimates for the outliers, which causes biased ATT estimates. In addition, RMSE for non-outlier observations is slightly higher due to higher residuals, sigma and non-outliers being forced into nodes with outliers. Uniquely, this response surface causes RMSE to increase with the number of iterations for default BARTZ.

The BARTZ estimates on the small dataset have more consistent estimates for the outliers as outliers must be estimated with non-outliers. However, the minimal node count limitation drastically increases the RMSE and the overall ATT compared to DBarts and BARTZ without such limitations.

Although the differences are marginal, it should be noted

Table 3. Performance Comparison of BART Models (Small Dataset)

	GPU BART	GPU BART (Restrictions Removed)	DBarts
RMSE	65.95	11.89	11.23
Estimate (Outl.)	-6.09	31.01	32.42
Estimate	23.16	5.32	5.05
Bias (Outl.)	-10.42	21.01	28.42
Bias	19.16	1.32	1.00

that removing the minimum node sizes sees small but negligible increases in runtime for this surface, although different response surfaces may vary.

5. Real World Testing

The original BARTZ paper was limited to Friedman Number 1 equation, which is a great benchmark but lacks some of the particular nuances of real-world data such as categorical variables, and extreme outliers. Thereby, other commonly used benchmarks were tested.

The main datasets tested were the Lalonde dataset, Infant Health and Development Program (IHDP) (Hill, 2011) and the ACIC 2016 (Dorie et al., 2019). The Lalonde Dataset (LaLonde, 1986) (Dehejia & Wahba, 1999) was created to test the effects of a jobs training program on an unbalanced number of individuals. The true ATT (Average Treatment on Treated) on a balanced dataset is $1794 dollars. However, the actual dataset had confounding variables as a far greater proportion of wealthy individuals were in the control group, as the wealthy had more opportunities to upskill, such as college or through familial connections but the treatment group was substantially less wealthy. Thus, simpler techniques would often be unable to separate the treatment effect from the confounders. For all real-world tests, BARTZ utilized a depth of 6.

The second of the datapoints was the semi-synthetic IHDP dataset, which utilized the real covariates and generated two differing synthetic response surfaces: one for the treatment and one for the control. An offset was further applied, generating a true ATE of 4. There were three surfaces: A, B and C, each with increasing complexity.

There was no substantial difference for the majority of the real-world tests (IHDP and Lalonde). It appears as if BARTZ was a marginally less precise in its predictions but the total amount of variation seems negligible. The limitation that leaf nodes must contain 5 observations appears to be the root cause of this imprecision.

Table 4. Performance Comparison: IHDP 100 seeds, Lalonde 50):

Dataset	Metric	BARTZ	DBarts
LaLonde	ATE	1643	1606
	Bias	-151	-188
	RMSE	NA	NA
	Coverage	1	1
IHDP Surface A	ATE	3.9606	3.9619
	Bias	-0.0394	-0.0381
	RMSE	0.2020	0.1964
	Coverage	0.95	0.96
IHDP Surface B	ATE	4.0954	4.1260
	Bias	0.0954	0.126
	RMSE	2.0490	2.002
	Coverage	0.83	0.87
IHDP Surface C	ATE	4.0394	4.0302
	Bias	0.0394	0.0302
	RMSE	2.6732	2.6260
	Coverage	0.83	0.86

The ACIC 2016 (Atlantic Causal Inference Competition), was a competition built around the Collaborative Perinatal Project dataset, wherein these covariates are used to produce complex response surfaces. We compared BartCause (Dorie & Hill, 2025) (underlying DBarts sampler) and Bartz to estimate the ATT (Average Treatment of Treated). The ACIC 2016 consisted of 77 settings, each setting having 100 unique seeds. The experiment tested on the first 20 seeds of each setting, thus running 1440 simulations.

Table 5. Performance Comparison of BART Variants on ACIC 2016 Setting 2

Metric	BartCause (DBarts)	BARTZ (No Limit)	BARTZ (Default)
RMSE	0.1513	0.1777	0.3916
Coverage	0.95	0.95	0.85
Bias	0.000875	0.0044	-0.0168

The vast majority of surfaces showed little variation between BARTZ and DBarts, with BARTZ only underperforming by the smallest of margins (RMSE about 7 percent worse and negligible differences in bias and coverage). However, some ACIC settings were substantially worse on BARTZ compared to DBarts, such as the 2nd setting. The second setting is unique as it is an exponential function, rescaled and offset, so that the treatment effect is the same for each observation. However, outliers exist for the independent variables, similar to the previous additive response surface devised. This causes the RMSE of default BARTZ to be substantially worse for Setting 2 compared to DBarts, unless

the node restrictions are fully removed. BARTZ was tested with a depth limitation of 6.

Table 6. ACIC: Setting 2 Results

Model	RMSE	Coverage	Bias
BartCause (DBarts)	0.1513	0.95	0.000875
BARTZ (Node Limit Removed)	0.1777	0.95	0.0044
BARTZ (Default)	0.3916	0.85	-0.0168

6. XOR Surface

One elegant but complex surface generally designed to stress statistical inference is the XOR (Exclusive OR) surface. This surface is difficult for inference algorithms due a discrete, branchy nature. If an odd number of statements are True, then the XOR gate responds with a True. However, if an even number of statements are True, then the statement computes False.

Perfectly representing an n-way XOR surface requires a decision tree of depth n, making it computationally demanding, as lower-level approximations would explicitly fail. We specifically utilized a 5-way interaction to stress-test these algorithms and to observe whether failure would occur. Furthermore, the first half of the dataset was used for training, while the second half for testing.

$$X_{i,j} \in \{0, 1, \ldots, 100\}, \quad i \in \{1, \ldots, 20000\}, \quad j \in \{1, \ldots, 7\} \tag{1}$$

$$\begin{aligned} b_0 &= I(X_0 < 50.5) \\ b_1 &= I(X_1 > 50.5) \\ b_2 &= I(X_2 < 50.5) \\ b_3 &= I(X_3 > 50.5) \\ b_4 &= I(X_4 > 50.5) \end{aligned}$$

$$Y = 10 \cdot \left(\sum_{k=0}^{4} b_k \pmod 2 \right) \tag{2}$$

These experiments utilized the exact same priors between all sets and the response surface appears like this:

Results: (It should be noted that I only had a single run with BARTZ)

It is challenging to isolate depth limitations or GROW-PRUNE limitations on performance slowdown. Limiting the depth of trees and limiting the sampler to grow-prune delay the convergence of the BART algorithm, with 4-step DBarts being substantially faster than the 2-step DBarts version and the two-step version being faster than BARTZ. Furthermore, limited tests show an association between the node-size limitation and substantive delays with respect to convergence.

Table 7. Experiment Results: RMSE Comparison

Experiment	RMSE
DBarts with Grow-Prune Only (50000 Burn-in, 10000 Samples)	4.83
DBarts with Grow-Prune Only (100000 Burn-in, 10000 Samples)	4.51
DBarts with Grow-Prune Only (200000 Burn-in, 10000 Samples)	0.91
DBarts with Default Sampler (50000 Burn-in, 10000 Samples)	1.46
GPU – BART (Default with 200k Burn-in, Depth of 8, 10K Samples)	2.35
GPU – BART (Default with 1M Burn-in, Depth of 8, 10K Samples)	0.86
GPU – BART (Default with 1M Burn-in, Depth of 10, 10K Samples)	0.84
GPU – BART (Default with 200k Burn-in, Depth of 12, 10K Samples)	1.32
GPU – BART (Default with 1M Burn-in, Depth of 12, 10K Samples)	0.92
GPU – BART (Node Limitations Removed, with 200k Burn-in, Depth of 8, 10K Samples)	3.11

6.1. Examining the Sum of Trees

A further investigation was necessary to determine the association between tree depth and sampler convergence, specifically using DBarts as a test. This experiment attempted series of procedures, calculating RMSE within a sample as each tree is updated and identifying the depth of trees that reduced RMSE. What is identified was between a strong correlation between tree depth and the absolute change in RMSE within a sample (0.6). Further tested is recording the RMSE and the mean depth of Trees for every 250 sample. Trees grew deeper, as more samplers were run.

The third procedure was iteratively removing trees from the ensemble and predicting with the remaining ensemble. Trees with a higher depth were better at capturing the response surface compared to shallower trees. The correlation between RMSE differentials when removed and the tree depth is 0.44.

A second experiment was run, wherein a single BART tree was made to estimate the outcome variable. We modified the priors of the BART algorithm to make deeper trees more

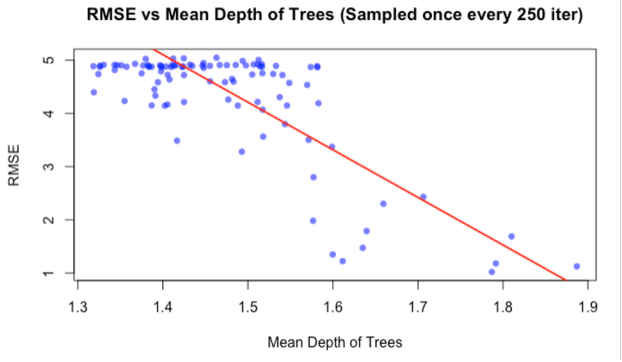


Figure 1. RMSE and Mean Depth of Trees.

likely, changing the base (alpha) and the power (beta) to 0.99 and 0.3 respectively. We attempt to run prior experiments on a single tree. The prior conclusions are further confirmed on the smaller surface, as we confirm an association between high tree depth and RMSE reduction.

We also compared MCMC steps to determine which moves were best at reducing RMSE. The moves are statistically distinct with a p-value of 7.82e-08 when an F-test is applied. Change seemed to cause the most substantive drop in RMSE. Prune was close, but PRUNE is not directly separable from GROW. Therefore, in this test, CHANGE appears to be absolutely better than other sampler moves with respect to convergence.

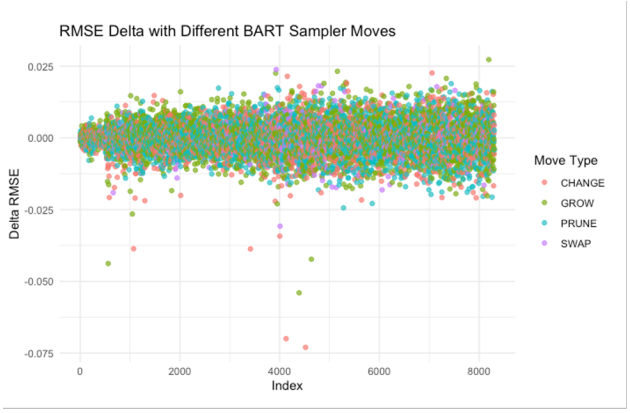


Figure 2. RMSE Change for All Moves.

Analysis of MCMC moves for One Tree (Last 50,000 Samples):

Although this response surface is contrived, it should be noted that there have been prior concerns about the usage of GROW-PRUNE, but concern generally apply to theoretical surfaces (Pratola, 2016). There may be potential to implement other sampler moves which may align better with the array-based architecture of BARTZ.

Table 8. MCMC Move Statistics and RMSE Impact

Move	RMSE Change	Occurrences
GROW	3.92×10^{-4}	3070
PRUNE	-3.11×10^{-4}	2988
SWAP	-3.11×10^{-5}	429
CHANGE	-5.84×10^{-4}	1820
MCMC Rejection	-1.39×10^{-5}	41693

7. Conclusion

BARTZ is extremely computationally powerful and appears generally applicable to almost all real-world datasets, with little difference in accuracy and precision compared to commonly used BART engines. There 5-observation node limitation mildly hampers the performance of BARTZ on small datasets and in instances of extreme outliers. In addition, BARTZ does have limitations with respect to discrete, non-approximatable, high-interaction surfaces as either the sampler takes far too long to converge or if the extreme increase in memory demands makes the mapping of such a surface computationally impractical. GPU-BART is a natural extension of BART into world of high-dimensional space and high-performance computing.

References

Breiman, L. Random forests. *Machine learning*, 45(1): 5–32, 2001.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. *Classification and Regression Trees*. CRC press, 1984.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794. ACM, 2016.

Chipman, H. A., George, E. I., and McCulloch, R. E. BART: Bayesian additive regression trees. *The Annals of Applied Statistics*, 4(1):266–298, 2010. doi: 10.1214/09-AOAS285.

Dehejia, R. H. and Wahba, S. Causal effects in nonexperimental studies: Reevaluating the evaluation of training programs. *Journal of the American Statistical Association*, 94(448):1053–1062, 1999.

Dorie, V. and Hill, J. *bartCause: Causal Inference using Bayesian Additive Regression Trees*, 2025. URL https://CRAN.R-project.org/package=bartCause. R package version 1.0-6.

Dorie, V., Hill, J., Shalit, U., Scott, M., and Cervone, D. Automated versus do-it-yourself methods for causal inference: Lessons learned from a data analysis competition. *Statistical Science*, 34(1):43–68, 2019.

Dorie, V., Chipman, H., and McCulloch, R. *dbarts: Discrete Bayesian Additive Regression Trees Sampler*, 2025. URL https://CRAN.R-project.org/package=dbarts. R package version 0.9-32.

He, J. and Hahn, P. R. Stochastic tree ensembles for regularized nonlinear regression. *Journal of the American Statistical Association*, 118(541):551–570, 2023. doi: 10.1080/01621459.2021.1942012.

Hill, J. L. Bayesian nonparametric modeling for causal inference. *Journal of Computational and Graphical Statistics*, 20(1):217–240, 2011.

LaLonde, R. J. Evaluating the econometric evaluations of training programs with experimental data. *The American Economic Review*, 76(4):604–620, 1986.

Liao, Y., Rubinsteyn, A., Power, R., and Li, J. Learning random forests on the gpu. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pp. 506–514, 2013.

Mitchell, R. and Frank, E. Accelerating the xgboost algorithm using gpu computing. *PeerJ Computer Science*, 3: e127, 2017.

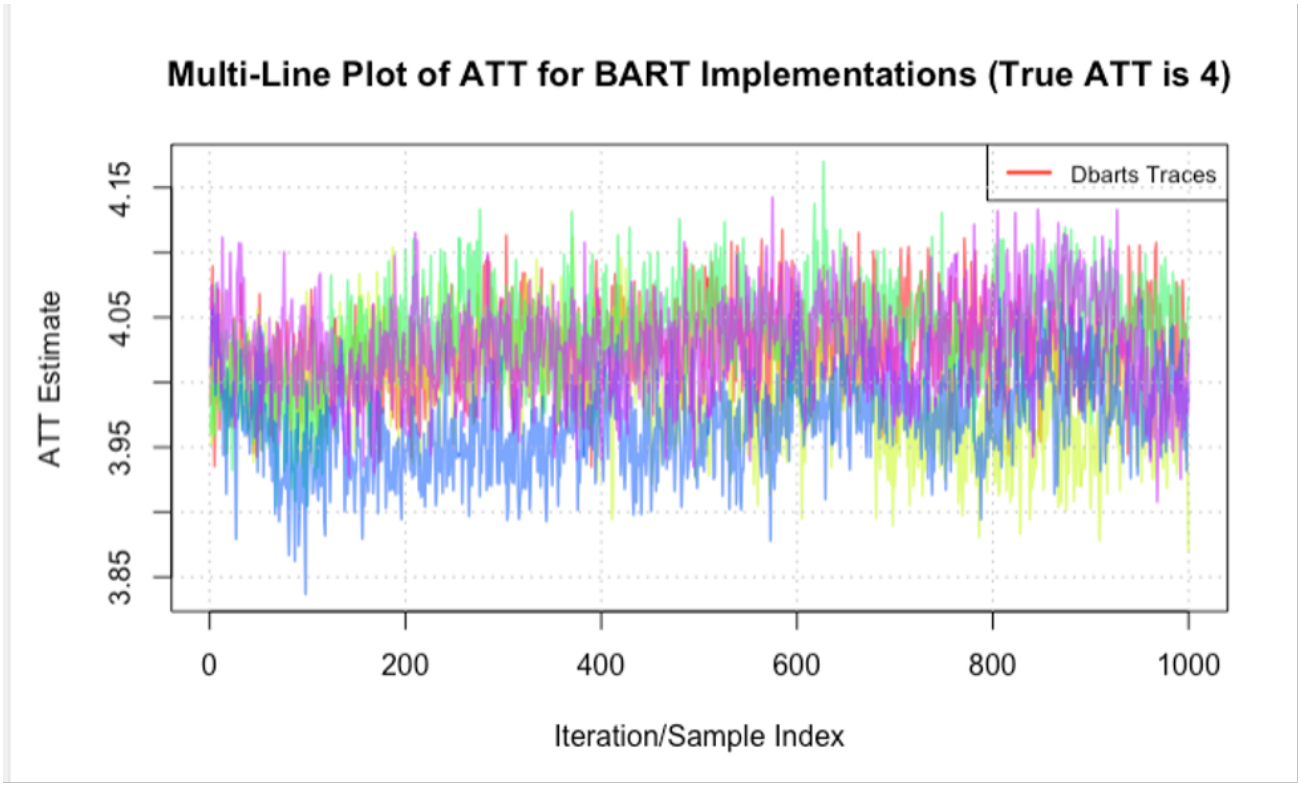
Pratola, M. T. Efficient metropolis–hastings proposal mechanisms for bayesian regression tree models. *Bayesian Analysis*, 11(3):885–911, 2016.

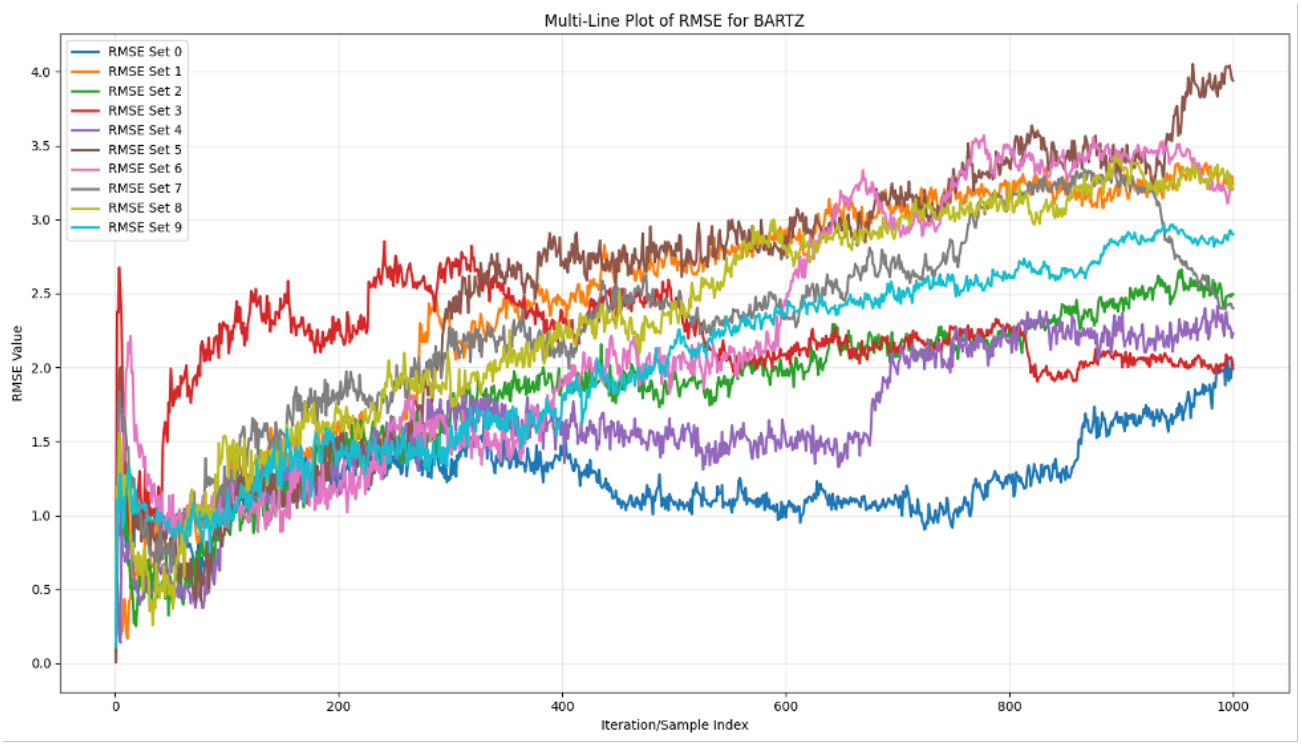
Pratola, M. T., Chipman, H. A., Gattiker, J. R., Higdon, D. M., McCulloch, R., and Rust, W. N. Parallel bayesian additive regression trees. *Journal of Computational and Graphical Statistics*, 23(3):830–852, 2014. doi: 10.1080/10618600.2013.842484.

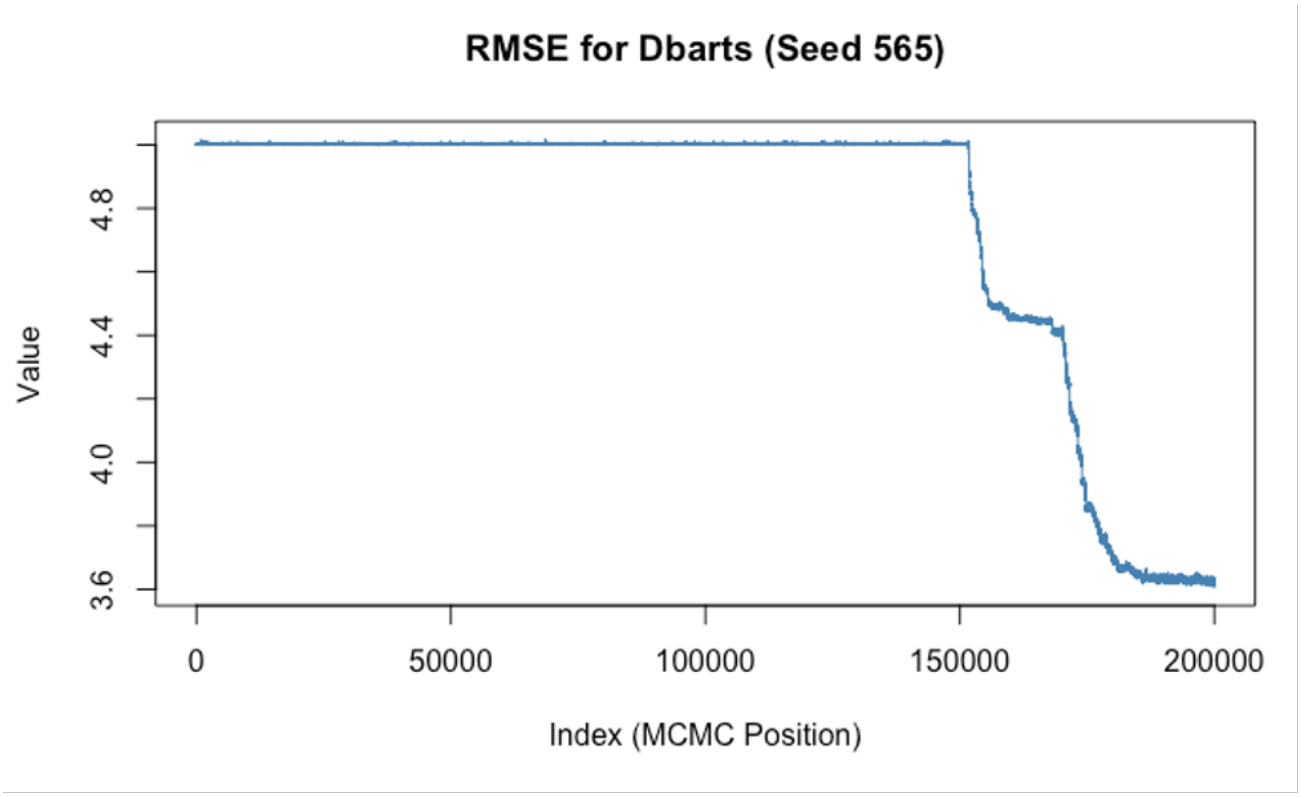
Sparapani, R., Spanbauer, C., and McCulloch, R. Nonparametric machine learning and efficient computation with bayesian additive regression trees: The bart r package. *Journal of Statistical Software*, 97 (1):1–66, 2021. doi: 10.18637/jss.v097.i01. URL https://www.jstatsoft.org/index.php/jss/article/view/v097i01.

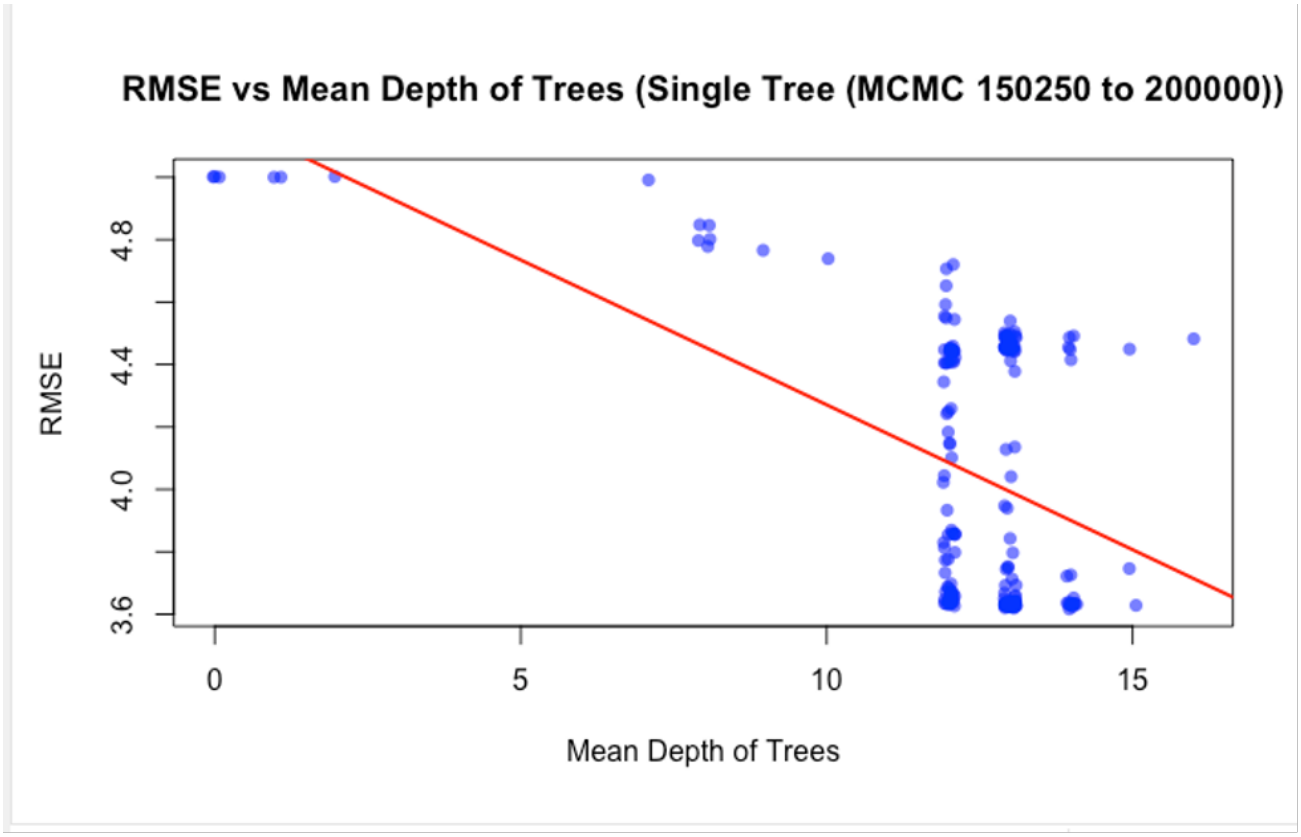
8. Appendix


Multi-Line Plot of ATT for BARTZ
ATT Value
Iteration/Sample Index
RMSE Set 0
RMSE Set 1
RMSE Set 2
RMSE Set 3
RMSE Set 4
RMSE Set 5
RMSE Set 6
RMSE Set 7
RMSE Set 8
RMSE Set 9
4.25
4.20
4.15
4.10
4.05
4.00
3.95
3.90
3.85
0
200
400
600
800
1000

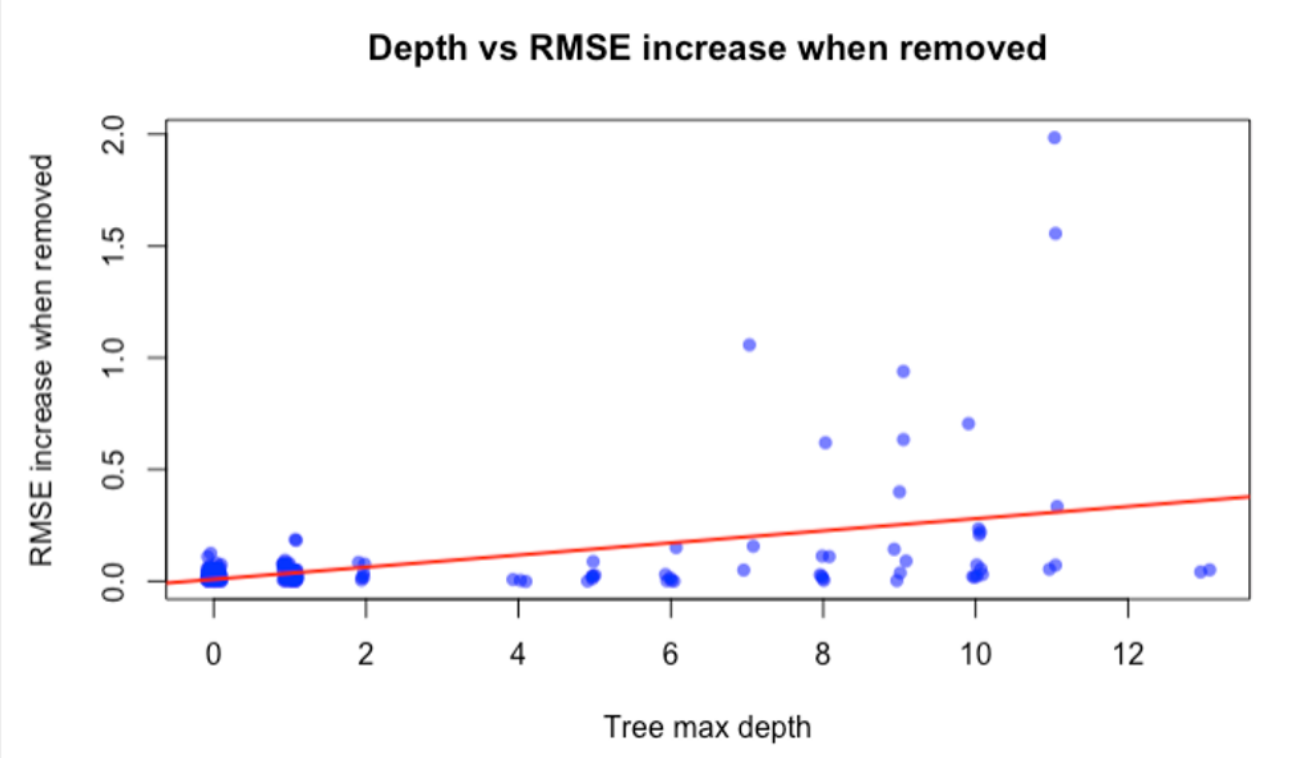

Multi-Line Plot of ATT for BART Implementations (True ATT is 4)
ATT Estimate
Iteration/Sample Index
Dbarts Traces
4.15
4.05
3.95
3.85
0
200
400
600
800
1000


Multi-Line Plot of RMSE for BARTZ
RMSE Set 0
RMSE Set 1
RMSE Set 2
RMSE Set 3
RMSE Set 4
RMSE Set 5
RMSE Set 6
RMSE Set 7
RMSE Set 8
RMSE Set 9
RMSE Value
Iteration/Sample Index
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
200
400
600
800
1000


RMSE for Dbarts (Seed 565)
Value
3.6
4.0
4.4
4.8
0
50000
100000
150000
200000
Index (MCMC Position)




```
                Df Sum Sq   Mean Sq F value   Pr(>F)
move_type        4 0.0014 0.0003378   9.692 7.82e-08 ***
Residuals    49995 1.7424 0.0000349
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```


Depth vs RMSE increase when removed
RMSE increase when removed
0.0
0.5
1.0
1.5
2.0
0
2
4
6
8
10
12
Tree max depth