# Application of Approximate Matrix Multiplication to Neural Networks and Distributed SLAM

Brian Plancher*
*Harvard University*
Cambridge, Massachusetts

Camelia D. Brumar*
*Worcester Polytechnic Institute*
Worcester, Massachusetts

Iulian Brumar*
*Harvard University*
Cambridge, Massachusetts

Lillian Pentecost*
*Harvard University*
Cambridge, Massachusetts

Saketh Rama*
*Harvard University*
Cambridge, Massachusetts

David Brooks
*Harvard University*
Cambridge, Massachusetts

***Abstract*—Computational efficiency is a critical constraint for a variety of cutting-edge real-time applications. In this work, we identify an opportunity to speed up the end-to-end runtime of two such compute bound applications by incorporating approximate linear algebra techniques. Particularly, we apply approximate matrix multiplication to artificial Neural Networks (NNs) for image classification and to the robotics problem of Distributed Simultaneous Localization and Mapping (DSLAM). Expanding upon recent sampling-based Monte Carlo approximation strategies for matrix multiplication, we develop updated theoretical bounds, and an adaptive error prediction strategy. We then apply these techniques in the context of NNs and DSLAM increasing the speed of both applications by 15-20% while maintaining a 97% classification accuracy for NNs running on the MNIST dataset and keeping the average robot position error under 1 meter (vs 0.32 meters for the exact solution). However, both applications experience variance in their results. This suggests that Monte Carlo matrix multiplication may be an effective technique to reduce the memory and computational burden of certain algorithms when used carefully, but more research is needed before these techniques can be widely used in practice.**

***Index Terms*—approximation, linear algebra, neural networks, robotics, SLAM**

## I. Introduction

In the past few decades there has been a large body of work focused on accelerating exact linear algebra kernels in hardware, motivating a range of inventions from GPU streaming multiprocessors to systolic arrays [1]. Since 2014, the dramatic proliferation of machine learning methods, particularly deep learning, has further increased the demand for efficient linear algebra operations while relaxing exactness requirements relative to traditional consumers of linear algebra in scientific computing [2]–[5].

Independently of trends in applied research, a surge of interest in approximation as a fundamental property of computational complexity has, in part, motivated a series of general approximation algorithms for fundamental linear algebra operations including matrix multiplication. One of the best known such proposal employs Monte Carlo sampling with replacement and offers asymptotic guarantees for the matrix norm of the resulting computation [6]–[8].

*All authors have contributed equally and are ordered alphabetically.

We observe that both the application domains of *deep learning* and *robotics* demand linear algebra operations but also tolerate some error in their results. This work builds on this intuition by applying and extending Monte Carlo methods for matrix multiplication to specific application domains and evaluating the resulting impact on end-to-end application speed and accuracy. Our applications of choice are neural networks for image classification and Distributed Simultaneous Localization and Mapping (DSLAM) for robotics, both of which rely heavily on matrix multiplications.

Focusing on matrix multiplication in particular, this work empirically evaluates the tightness of bounds in the algorithm for sampling with replacement, which is an attractive solution due to previously-verified theoretical bounds [6]. We explore the limitations of the existing algorithm, propose modifications, and develop a corresponding error prediction model using computed error bounds. This version of Monte Carlo matrix multiplication is then applied in the context of our target applications in order to evaluate the practicality and potential performance improvements of theoretical results for interesting end-to-end, compute-constrained problems.

## II. Related Work

Simplified approaches to linear algebra can have a significant impact on computational overhead and memory requirements for critical applications, and have been studied since the inception of factor models in psychology [9], [10]. More recently, a series of theoretical works have set compelling bounds on Monte Carlo algorithms for a set of linear algebra operations, including matrix multiplication, low-rank approximation, and matrix decomposition [6]–[8]. Similar randomized algorithms have been proposed for low-rank matrix factorizations, such as singular value and QR decomposition [11].

Many alternate approaches to approximate matrix multiplication exist. For example, one such proposal leverages Fast Fourier Transforms (FFTs) and treats matrix multiplication as a low-rank polynomial multiplication [12]. Another approach is based on random projections [13]. Finally, another family of approaches satisfies additional constraints, such as retaining a subset of columns unchanged in the approximation, or by

weakening a full low-rank approximation to only be locally low-rank over a matrix [14], [15].

This work focuses on the approximation of matrix multiplication using Monte Carlo methods. Monte Carlo methods are particularly appealing from a system performance perspective due to the ability to approximate an output without having to compute (or even fetch from main memory) the full input matrices in the context of matrix multiplication. A paper adopting a strategy similar to the one explored in this paper characterizes the applicability of this to linear regression directly [16] and similar approximation strategies have been proposed for other areas of machine learning, such as kernel methods [17] and clustering [18]–[21]. However, application-focused studies in other domains are limited and little work has been done to analyze the impact of Monte Carlo methods on end-to-end application performance either in terms of accuracy (as defined in a given domain) or measured run time improvements. We address both of these issues in this work.

## III. Application Background

### A. Neural Networks

Neural Networks (NNs) are becoming increasingly established as a state-of-the-art approach for a multitude of cutting-edge classification tasks, including object detection and tracking, speech recognition, and translation [22]. It is well known that NNs are reasonably fault-tolerant and resilient to approximations such as reduced numerical precision, pruning, and architectural faults (e.g., flipped bits in stored values) [3], [4], [23], [24]. The computational workhorse of NNs is matrix multiplication, and accelerating the execution of NN inference by supporting faster matrix multiplication is an incredibly fertile area of research [2], [22], [25]–[27]. Thus, applying approximate matrix multiplication techniques for NN inference is a promising prospect for reducing the overall computational workload for critical applications.

### B. Distributed Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is the process by which a robot builds a map of its environment while simultaneously localizing itself in this map [28], [29]. A common approach to computing the current belief of the map and robot state is with an *Extended Kalman Filter (EKF)* [30]. The EKF computes a linearization of both the robot's *motion* and *sensor measurement*, and then computes the maximum likelihood estimate of the belief. This process is repeated at discrete time steps as the robot moves through the environment. As such, SLAM iteratively refines noisy estimates of the map and robot location through future (noisy) sensor readings.

Recently, Distributed Simultaneous Localization and Mapping (DSLAM) [31], [32] was proposed to speed up this process by leveraging a swarm of $N$ robots to collectively complete this task over $T$ timesteps. The DSLAM problem considers the state vector $X$ as the concatenation of the states of *all* $N$ robots $x \in \mathcal{R}^n$ whose motion is defined as $f(\cdot)$ and the position of $M$ landmarks $l \in \mathcal{R}^m$ in the environment that are found with a sensor measurement function $h(\cdot)$. Similarly it considers $U$ as the concatenation of all control inputs $u$, $Y$ as the concatenation of all sensor readings $y$, and $\Sigma$ as the *full covariance matrix* between *all* the state and landmark positions. It then computes the updated state $X_t$ at time $t$ as shown in Algorithm 1 (where $Q$ and $R$ represent the process and measurement noise respectively).

**Algorithm 1** DSLAM

$X_0, \Sigma_0 \leftarrow X_{init}, \Sigma_{init}$
**for** $i = 1 \dots T$ **do**
  **Motion Update**
  $X_{t|t-1} = f(X_{t-1}, U_t)$
  $F = \frac{\partial f(X_{t-1}, U_t)}{\partial X_{t-1}}$
  $\Sigma_{t|t-1} = F\Sigma_{t-1}F^T + Q_t$
  **Measurement Update**
  $y_t = h(X_{t-1})$
  $y_{t|t-1} = h(X_{t|t-1})$
  $H = \frac{\partial h(X_{t-1})}{\partial X_{t-1}}$
  $S = H\Sigma_{t|t-1}H^T + R_t$
  $K = \Sigma_{t|t-1}H^T S^{-1}$
  $X_t = X_{t|t-1} + K(y_t - y_{t|t-1})$
  $\Sigma_t = (I - KH)\Sigma_{t|t-1}$
**end for**

In the DSLAM algorithm, we note that the covariance matrix $\Sigma \in \mathcal{R}^{(Nn+Mm)\times(Nn+Mm)}$ grows quadratically with the number of, and state dimensions of, both the robots and landmarks. Therefore, for large numbers of robots and/or landmarks, and/or for more complex robots, there is ample opportunity for approximation to increase the performance of the algorithm by accelerating the large, computationally intensive matrix multiplication operations.

## IV. Monte Carlo Matrix Multiplication (MCMM)

### A. Background

The basic cubic-time matrix multiplication algorithm for $P = AB = A \times B$ produces each element in the output matrix $P$ as $P_i^j = A_i \times B^j$, where $P_i^j$, $A_i$, and $B^j$ denote respectively element $(i, j)$ of $P$, the $i$-th row of $A$, and the $j$-th column of $B$. A variety of approximation strategies for matrix multiplication reformulate matrix multiplication in terms of outer products, as shown in Equation 1. Note that multiplying $A^i$ times $B_i$ for a particular $i$, results in a partial result which has the same dimensions as the final matrix $P = A \times B$.

$$A \times B = \sum_{i=0}^{n} A^i B_i \tag{1}$$

Randomized algorithms sample these row-column pairs [11]. The sampling involves picking $c$ columns of A and $c$ rows of B according to a random distribution. The matrices C and R are then built with the $c$ row-column pairs and regularized in order to account for missing rows and columns as shown in Equation 2.

$$C^t = \frac{A^{i_t}}{\sqrt{c * p_{i_t}}} \qquad R_t = \frac{B_{i_t}}{\sqrt{c * p_{i_t}}} \tag{2}$$

$C$ and $R$ are then multiplied together to produce the final result. In this paper, we build on Monte Carlo Matrix Multiplication [6], a recently proposed algorithm that follows this approach.

Most approximate computing proposals such as Loop Perforation [33], Task Skipping [34] or Approximate Task Memoization [35] rely on profiling and program training of inputs to empirically determine the overall program error when using real test inputs. However, if the training inputs are not sufficiently representative of real world cases, the approximation mechanism might lead to higher unexpected errors.

MCMM, on the other hand, allows the user to determine if the approximate matrix multiplication is suitable for the problem encoded in the operand matrices *without having to run the full multiplication on training inputs and with strong theoretical guarantees* as defined in Equation 3.

$$\|AB - CR\| = O\left(\frac{\|A\| * \|B\|}{\sqrt{c}}\right) \tag{3}$$

Importantly, $c$ does not refer to a percentage of the row-column pairs sampled from the input matrices A and B, but to the absolute number. This means that if the matrices A and B are large enough, even with a low sampling rate, the denominator will be large, leading to a relatively low error bound. That said, larger matricies will also have larger norms (in this work, all matrix norms are Frobenius norms unless otherwise specified).

It is important to note that this bound only holds when the sampling is done with replacement. Somewhat counter-intuitively, this means that we may sample a particular row-column pair multiple times and never sample others.

### B. Limitations

Unfortunately, these guarantees are insufficient from an application perspective for several reasons:

1) The provided error bound is *absolute*, not *relative to the ground truth*. This means that we can bound the error to a small absolute error but still have a very significant relative error compared to the norm of the resulting matrix, where the relative error bound is given by:

$$O\left(\frac{\|A\| * \|B\|}{\sqrt{c} * \|AB\|}\right) \tag{4}$$

2) The provided error bound is expressed using asymptotic notation. This result is weaker than a hard error bound curve depending on some feature of the operand matrices, e.g., the matrix norms.
3) To take advantage of the performance advantage of approximation while utilizing an error bound as a guide, we want to compute a relative error bound without requiring the exact result $AB$.
4) Sampling in [6] is performed with replacement with non-uniform probabilities for each of the row-column pairs of the input matrices, but the exact choice of weights is completely left to the user.

### C. Improved Error Prediction

We construct a model to predict the MCMM relative error with high accuracy. We empirically show that the expression

$$\frac{\|A\| * \|B\|}{\sqrt{c} * \|AB\|} \tag{5}$$

is highly correlated with the real relative error between the exact matrix multiply and the MCMM solution:

$$\frac{\|AB - CR\|}{\|AB\|} \tag{6}$$

The asymptotic notation in Equation 4 can be expressed as,

$$\frac{\|AB - CR\|}{\|AB\|} \leq factor * \frac{\|A\| * \|B\|}{\sqrt{c} * \|AB\|}$$

which says that by multiplying the expression (Equation 4) by a constant factor, the exact error will be bounded. Intuitively, the norm of the approximate result $CR$ should be similar to the norm of $AB$, even though the individual elements might differ. In Section V, we show that:

$$\frac{\|A\| * \|B\|}{\sqrt{c} * \|CR\|} \tag{7}$$

is highly correlated with, and as such can be used to predict, the real relative error (Equation 6).

### D. Improved Sampling Methods

We propose a sampling strategy to enable highly predictable errors across many example matrices. This allows for an adaptive approach where MCMM is performed for error-tolerant computations and approximation is avoided for less resilient computations. The intuition for our strategy is that if the norms of column $A^i$ and row $B_j$ are relatively small, their multiplication will yield a small norm and have a smaller impact in the final result. We assign probabilities:

$$p_i = \frac{w_i}{\sum_{j=1}^{n} w_j} \qquad w_i = \|A^i\| * \|B_i\| \tag{8}$$

to each of the row-column pairs. Section V demonstrates that this approach yields higher accuracy than uniform sampling and leads to more predictable error bounds using Equation 7.

## V. High-Level MCMM Evaluation

### A. Methodology for local MCMM accuracy analysis

In order to analyze the local accuracy of different sampling strategies and error prediction for MCMM, we test our implementation on matrices from many different domains from the Florida Sparse matrix library [36]. For each example, we compute the multiplication of a given matrix A by the transpose of A ($A^T$).[1]

[1]The Florida Sparse Matrix repository does not give pairs of matrices for each particular problem. Therefore, using this data alone may impose limitations on the effectiveness of our strategy. However, we note that this operation ($C = A^T A$) occurs in many applications. For example, when a matrix that needs to be inverted has more rows than columns, e.g. when solving an over-determined system, multiplying $A^T$ by $A$ results in an invertible square (symmetric) matrix.

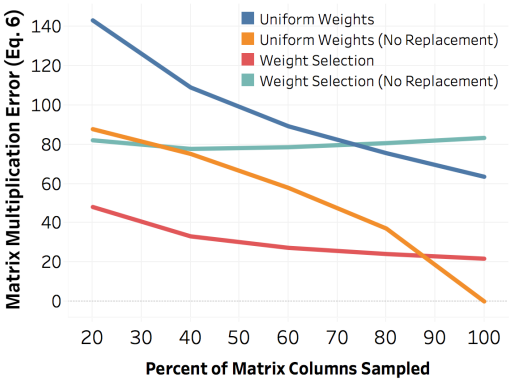


Fig. 1. Average error in output matrix for different sampling strategies over all matrices. The x-axis represents the percentage of sampled row-columns, and the y-axis is Equation 6.

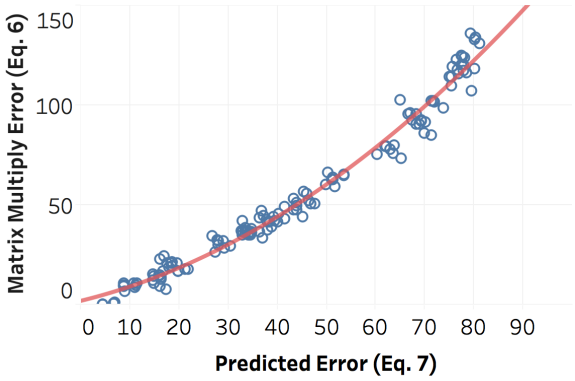


Fig. 2. The relation between the predicted MCMM relative error (Eq. 7) and the real matrix multiply error (Eq. 6). Each point represents sampling 40% of the row-column pairs of the Florida Sparse Matrices. We use least squares (shown in red) to model the tight correlation between the Expression 6 and the real matrix multiply error since replacing $||AB||$ with $||CR||$ requires some correction of the error bound formula as described in Subsection IV-C.

We use a C++ MCMM implementation built with the Eigen [37] library to perform the matrix-matrix multiplications for selected matrices.[2] These examples exhibit a total execution time between 10ms and 500ms on an Intel® Core™ i5-7440HQ CPU @ 2.80GHz.

### B. Evaluation of sampling strategies

We empirically evaluate the usefulness of four different sampling approaches for MCMM row-column sampling: with vs. without replacement and with uniform weights (UW) vs. non-uniform weights (NUW), as shown in Figure 1.

In almost all cases NUW + Replacement has the lowest error. Interestingly this is the only strategy satisfying the theoretical requirements described in Section IV. However, sampling with replacement always results in an inherently approximate MCMM as it may result in re-sampling particular row-column pairs and never sampling others.

The UW + No Replacement strategy overcomes this limitation for 100% sampling at the cost of robustness for sampling lower than 80%. Unfortunately, this benefit is lost if weights are non-uniform (NUW + No Replacement) as Equation 2 divides the partial product result by the probability of sampling to account for potentially non-sampled row-column pairs, which is useful only when we sample with replacement.

### C. MCMM error prediction

A benefit of our approach is that we can tightly control the error of each specific approximate computation due to the theoretical guarantees of MCMM. We derive a model for the predicted error by plotting the actual error against the predicted error under the theoretical error bound (Equation 5) and noticing a highly accurately linear fit to $y = x$. However, as mentioned in Section IV, computing this bound requires computing the full $AB$ matrix and is therefore impractical. Instead, we examine the actual error against our improved predicted error using the $CR$ matrix in place of the $AB$ matrix (Equation 7). While this no longer fits the line $y = x$, we found that when sampling with our improved sampling method, NUW + Replacement, it did produce a predictable trend, as shown in Figure 2. We fit a quadratic model to this data (shown in red) and found the best least squares fit was:

$$0.01x^2 + 0.6x + 1.52 \text{ , where } x = \frac{\|A\| * \|B\|}{\sqrt{c} * \|CR\|} \quad (9)$$

Equation 9 can be used to discard approximate matrix computations that produce unacceptable predicted errors. Based on this prediction formula, application developers can design an adaptive algorithm that manipulates the percentage of sampled data until the predicted error is below an acceptable threshold using the approximate result $CR$.

## VI. Domain-Specific Case Studies

### A. Neural Network Inference

We apply the techniques discussed in Section V to two NNs for the well-known task of optical digit recognition with the MNIST dataset [38]. A specific bottleneck of NN execution is fetching weight values from memory, and MCMM provides an additional advantage of reducing overall memory bandwidth requirements by requiring only a subset of values per layer, thus reducing the working set size of the computation in addition to enabling more efficient computation.

Table I gives the baseline parameters of the two pre-trained NNs we consider. Both models approach state-of-the-art accuracy (less than 2% baseline classification error) and their weight matrices are relatively sparse (up to 90% zero-valued). MNIST-FC is comprised of three fully-connected layers with weight matrix sizes up to $1000 \times 300$. MNIST-CNN is a LeNet5 model with two convolutional layers followed by two fully-connected layers for final classification.

[2]The specific matrices used were: $bp\_0$, $orbitRaising\_1$, $GD00\_c$, $fs\_541\_2$, $mbeacxc$, $oscil\_dcop\_01$, $bfwa\_398$, $bfwa\_782$, $west\_0381$, $nos1$, $nos7$, $DK01r$, $GRE\_1107$,$BCSSTK\_10$, $BCSSTK\_12$, $BCSSTK\_27$, $BCSSTM27$, $nnc1374$, $coater1$, $rail\_1357$, $model4$, $freeFlyingRobot\_2$, $spaceStation\_10$, $spaceStation\_11$, $spaceStation\_12$, $mahindas$, $pores\_2$, $lp\_ganges$, $fpga\_dcop\_01$, $rdb\_1250$, $cz\_1268$

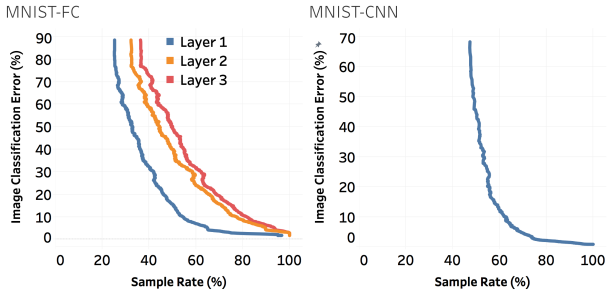


Fig. 3. Average image classification error for Fully-Connected (MNIST-FC, left) and Convolutional (MNIST-CNN, right) NN layers and corresponding rate of sampling. To maintain 97% classification accuracy, only the first layer in MNIST-FC should be approximated (sample rate 76%), while both convolutional layers of MNIST-CNN can be approximated (sample rate 82%).

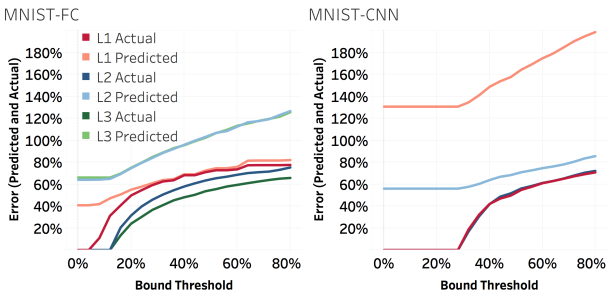


Fig. 4. We compare the actual error (approximate result vs. correct output) and the predicted error (theoretical bound) for different NN layers as we vary the bound threshold, as tested using Equation 9.

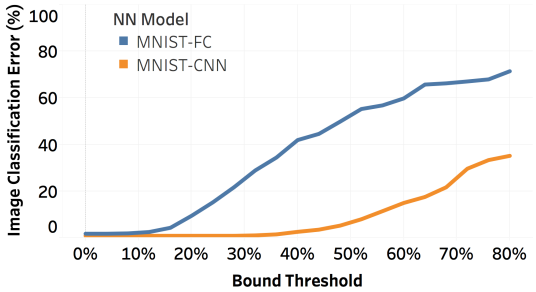


Fig. 5. Varying bound threshold leads to graceful degradation in end-to-end image classification accuracy for both NNs considered.

MCMM is applied to batched inference for fully-connected layers in MNIST-FC and sparsified convolutional layers in MNIST-CNN. We vary the threshold against which Equation 9 is compared, employing MCMM with replacement for the weighted sampling technique described in Section V when the predicted error does not exceed the given threshold. We report the average acceptable sampling rate per NN layer and the resulting image classification error across the entire MNIST test set for 100 trials per threshold value in Figure 3. Note that 100% sampling rate corresponds to no approximation being conducted (i.e., if no acceptable sample rate under 100% is determined, the calculation is performed exactly).

Layer 1 of MNIST-FC is the largest of the fully-connected layers considered, and we see an initially gradual degradation in the image classification accuracy at sample rates down to about 65%. The abrupt degradation of image classification accuracy towards random classification (90% error for 10 possible classifications) at lower sampling rates is characteristic of the behavior of NNs under other approximation schemes [4]. Reducing the number of required row-column pairs for the largest FC layer of MNIST-FC to a sampling rate of 76% would reduce the overall number of parameters fetched from memory for a given inference by over 20% and reduces the execution time by about 15-20% while maintaining 97% classification accuracy, which suggests that MCMM is a compelling technique for reducing the computational and memory bandwidth required for NN inference.

To validate the findings of Section V-C against the specific matrix multiplication operations within the NNs, we observe the discrepancies between the predicted error (Equation 4) and the actual error (Equation 6) for a given bound threshold, summarized in Figure 4. For both convolutional layers in MNIST-CNN and for layers 2 and 3 in MNIST-FC, the predicted error bounds are far too conservative compared with the actual error. However, Layer 1 of MNIST-FC exhibits very close agreement between the actual and predicted errors, and we note that this is also the layer we found to be most amenable to approximation.

TABLE I
BASELINE CONFIGURATION OF THE NNS AND THE MINIMUM POSSIBLE % SAMPLE RATE THAT MAINTAINS 97% ACCURACY ('SAMPLE RATE').

| Model | Baseline Error | Zero-Valued | Sample Rate |
|---|---|---|---|
| MNIST-FC | 1.4% | 84.9% | 76% |
| MNIST-CNN | 0.8% | 89.9% | 82% |

The adaptive strategy for determining acceptable sample rates per layer of the NN computation is particularly advantageous because we find that the bound threshold across all layers correlates to the actual application accuracy (i.e., image classification error). Figure 5 explicitly compares the relationship between the bound threshold and the image classification error for both NNs. We note that compared to varying the sampling rate in isolation, the adaptive strategy of imposing a varying bound threshold leads to a gradual degradation in image classification accuracy for both models considered. This suggests that MCMM with error prediction offers a direct trade-off between approximation and final application accuracy, in addition to verifying that the bounds developed in Section V-C can be an effective indicator of application accuracy for this particular example.

### B. DSLAM

Next, we apply the techniques discussed in Section V to the DSLAM problem. For our experiments, we consider a two-dimensional DSLAM environment with $N = 25$ simple point robots whose state $x$ is simply their 2D position $(r_x, r_y)$ and orientation $\theta$ under random motion. $M = 20$ landmarks are used, which similarly have 2D positions. We assume that each robot has a GPS sensor, which will relay a noisy reading of the robot's 2D position, and a monocular camera, which will relay a noisy reading of the angle $\alpha$ and distance $d$ to the

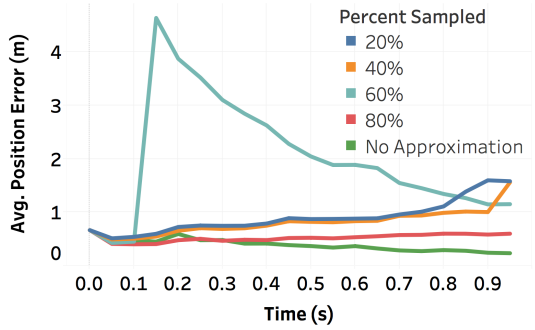


Fig. 6. Error in position estimations over time averaged over 10 trials for DSLAM under various levels of approximation.

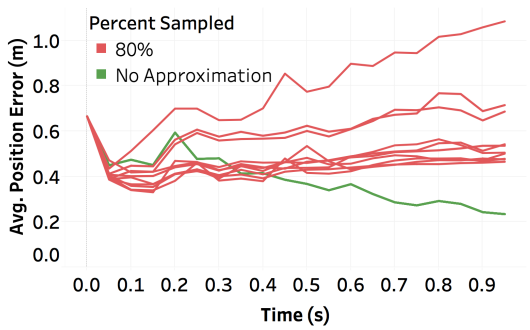


Fig. 7. Average position error over time results per trial at 80% sampling (red) compared to DSLAM without approximation (green).

nearest landmark within range $d_{max}$ meters and within offset angle $\alpha_{max}$ or no reading at all. We apply $T = 20$ random control steps to each robot with a total exploration time of 1 second in a grid world of size 20 by 20 meters starting from a random initial configuration.

We ran 10 trials across 5 levels of approximation (no approximation, 80% sampling, 60% sampling, 40% sampling, 20% sampling) using our improved sampling method, NUW + Replacement. We find that by only approximating the two most expensive steps in the computation (the matrix multiplications involving $\Sigma$ in lines 5 and 12 in Algorithm 1), we achieve 1.15X to 1.83X speedups in the end-to-end computation as shown in Table II.

However, we find that the end-to-end accuracy of the DSLAM algorithm degrades with increased sampling, as shown in Figure 6. In some trials, the random nature of the sampling may induce large errors in the computation. This is best seen in the large jump in error in the 60% sampling case (shown in teal) between 0.1 and 0.2 seconds. We also see this in Figure 7, which shows the variance in output from 10 trials at 80% sampling. Even for the most accurate trial, we find sampling does not converge to the baseline solution and has at least twice as much average error. However, with only 20% sampling the increased error is still on average only on the order of meters, which may acceptable for some applications.

Unfortunately, when attempting to apply the adaptive strategy to DSLAM computations, the predicted errors were on the order of 200%, which never leads to the use of our approximation strategies. We believe this may be due to the particular structure of the matrices used in this computation and hope to further explore this issue in future work.

TABLE II
DSLAM EKF SPEEDUP UNDER VARIOUS LEVELS OF APPROXIMATION.

| Percent Sampled | Relative Speedup | Std. Dev. of Speedup |
|---|---|---|
| Exact | 1 | N/A |
| 80% | 1.15 | 0.075 |
| 60% | 1.33 | 0.12 |
| 40% | 1.54 | 0.12 |
| 20% | 1.83 | 0.25 |

## VII. Discussion and Future Work

There is a large space to explore in considering the relationship between application error and theoretical guarantees with approximate computation. In applying MCMM to NN inference, we find certain operations within the NN are more amenable to this form of approximation, though application accuracy does degrade significantly below sampling rates of about 75%. However, it is remarkable and encouraging that actual error for the largest layer is similar to the MCMM predicted error, which is given by the theoretical guarantees.

For the DSLAM use case, there are further research opportunities in terms of closing the gap between the predicted error and the actual error. We note that this gap may be driven by over-fitting in our predicted error model (Equation 9). We leave for future work more general approaches to computing predicted errors and those that leverage more heterogeneous matrices.

We also leave for future work other linear algebra operations amenable to approximation such as finding eigenvalues and eigenvectors, solving systems of equations, and performing low rank matrix approximation.

## VIII. Conclusion

To achieve disciplined approximate computation in several end-to-end applications, we divide the problem into two steps. First, we expand on MCMM's formal error bounds to provide a fast relative error formula that can be used to predict the error in the local approximate matrix multiplications. Second, we characterize the relationship between the predictable local error and the end-to-end application error. Using this approach, we find promising results for DNNs and DSLAM that highlight the need for future work developing error prediction models to reduce the variance in the results.

## Acknowledgments

This work was partially supported by the Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA, the DARPA DSSoC program, and the National Science Foundation Graduate Research Fellowship (under grant DGE1745303). Any opinions, findings, conclusions, or recommendations expressed in this material

are those of the authors and do not necessarily reflect those of the funding organizations.

The authors would like to thank Moritz Graule and Patrick Varin for help with DSLAM and are grateful to the anonymous reviewers for their comments and suggestions.

## REFERENCES

[1] H. Kung and C. E. Leiserson, "Systolic arrays (for vlsi)," in *Sparse Matrix Proceedings 1978*, vol. 1. Society for Industrial and Applied Mathematics, 1979, pp. 256–282.

[2] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: ACM, 2017, pp. 1–12.

[3] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernandez-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *ISCA*, 2016.

[4] B. Reagen, L. Pentecost, U. Gupta, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei, "Ares: A framework for quantifying the resilience of deep neural networks." in *2018 The 55th Annual Design Automation Conference (DAC)*, June 2018.

[5] G. Li, S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *SC*, 2017.

[6] P. Drineas, R. Kannan, and M. W. Mahoney, "Fast monte carlo algorithms for matrices i: Approximating matrix multiplication," *SIAM Journal on Computing*, vol. 36, no. 1, pp. 132–157, 2006.

[7] ——, "Fast monte carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix," *SIAM Journal on computing*, vol. 36, no. 1, pp. 158–183, 2006.

[8] ——, "Fast monte carlo algorithms for matrices iii: Computing a compressed approximate matrix decomposition," *SIAM Journal on Computing*, vol. 36, no. 1, pp. 184–206, 2006.

[9] A. S. Householder and G. Young, "Matrix approximation and latent roots," *The American Mathematical Monthly*, vol. 45, no. 3, pp. 165–171, 1938.

[10] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.

[11] N. Halko, P.-G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM review*, vol. 53, no. 2, pp. 217–288, 2011.

[12] R. Pagh, "Compressed matrix multiplication," *ACM Transactions on Computation Theory (TOCT)*, vol. 5, no. 3, p. 9, 2013.

[13] T. Sarlos, "Improved approximation algorithms for large matrices via random projections," in *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*. IEEE, 2006, pp. 143–152.

[14] G. H. Golub, A. Hoffman, and G. W. Stewart, "A generalization of the eckart-young-mirsky matrix approximation theorem," *Linear Algebra and its applications*, vol. 88, pp. 317–327, 1987.

[15] J. Lee, S. Kim, G. Lebanon, and Y. Singer, "Local low-rank matrix approximation," in *International conference on machine learning*, 2013, pp. 82–90.

[16] M. B. Cohen, Y. T. Lee, C. Musco, C. Musco, R. Peng, and A. Sidford, "Uniform sampling for matrix approximation," in *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*. ACM, 2015, pp. 181–190.

[17] A. J. Smola and B. Schölkopf, "Sparse greedy matrix approximation for machine learning," 2000.

[18] A. Deshpande, L. Rademacher, S. Vempala, and G. Wang, "Matrix approximation and projective clustering via volume sampling," in *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. Society for Industrial and Applied Mathematics, 2006, pp. 1117–1126.

[19] A. Deshpande and S. Vempala, "Adaptive sampling and fast low-rank matrix approximation," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2006, pp. 292–303.

[20] A. Banerjee, I. Dhillon, J. Ghosh, S. Merugu, and D. S. Modha, "A generalized maximum entropy approach to bregman co-clustering and matrix approximation," *Journal of Machine Learning Research*, vol. 8, no. Aug, pp. 1919–1986, 2007.

[21] M. B. Cohen, S. Elder, C. Musco, C. Musco, and M. Persu, "Dimensionality reduction for k-means clustering and low rank approximation," in *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. ACM, 2015, pp. 163–172.

[22] B. Reagen, R. Adolf, P. Whatmough, G.-Y. Wei, and D. Brooks, "Deep learning for computer architects," *Morgan & Claypool Synthesis Lectures on Computer Architecture*, 2017.

[23] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015. [Online]. Available: http://arxiv.org/abs/1510.00149

[24] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, pp. 1737–1746.

[25] M. Adelman and M. Silberstein, "Faster neural network training with approximate tensor operations," *arXiv preprint arXiv:1805.08079*, 2018.

[26] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, 2014.

[27] Y. Guo, A. Yao, H. Zhao, and Y. Chen, "Network sketching: Exploiting binary structure in deep cnns," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5955–5963.

[28] B. Tim and H. Durrant-Whyte, "Simultaneous localization and mapping (slam): Part i," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[29] ——, "Simultaneous localization and mapping (slam): Part ii," *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[30] L. Ljung, "Asymptotic behavior of the extended kalman filter as a parameter estimator for linear systems," *IEEE Transactions on Automatic Control*, vol. 24, no. 1, pp. 36–50, 1979.

[31] "Distributed simultaneous localization and mapping for mobile robot networks via hybrid dynamic belief propagation," *International Journal of Distributed Sensor Networks*, vol. 13, no. 8, 2017.

[32] S. Thrun, "Simultaneous localization and mapping," in *Robotics and cognitive approaches to spatial mapping*. Springer, 2007, pp. 13–41.

[33] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 124–134.

[34] M. Rinard, "Probabilistic accuracy bounds for fault-tolerant computations that discard tasks," in *Proceedings of the 20th annual international conference on Supercomputing*. ACM, 2006, pp. 324–334.

[35] I. Brumar, M. Casas, M. Moreto, M. Valero, and G. S. Sohi, "Atm: Approximate task memoization in the runtime system," in *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*. IEEE, 2017, pp. 1140–1150.

[36] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1:1–1:25, Dec. 2011.

[37] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[38] Y. LeCun and C. Cortes, "The MNIST database of handwritten digits," http://yann.lecun.com/exdb/mnist/.