Research Statement

Ankur Mallick

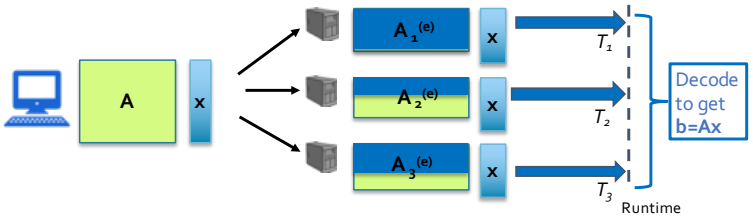
My goal is to enable *fault tolerant distributed computing* in a post-Moore's Law world. Fault tolerance is required in all large-scale computing systems where multiple nodes perform computations, and hardware/software/power failures can cause some nodes to slow down or fail resulting in delayed or incomplete computations. Historically, redundancy via task replication, and dynamic recovery via fault detection and task reallocation [1] were the prevalent approaches for fault tolerance. The centralized monitoring and data movement required for dynamic recovery is generally too expensive for modern cloud computing frameworks, which have therefore largely relied on redundancy for fault tolerance [2, 3]. However adding redundancy via replication requires a significant amount of spare computing capacity which may not be available anymore due to the end of Moore's law and the saturation of computing speeds. Thus there is a strong need for novel, *low-cost* and *resource-efficient* approaches for fault tolerance.

My Research. The main thrusts of my research are summarized below. Each provides a novel perspective on fault tolerance and introduces a solution that can operate within the cost and resource constraints of modern cloud computing settings.

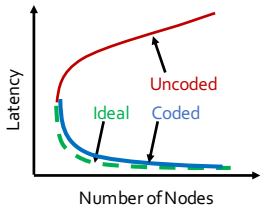
1. **Algorithm based fault tolerance using erasure codes** [4]: This work builds fault tolerance *directly* into the computation by identifying mathematical functions (for e.g. linear mappings) of the original computation such that it can be recovered from the results of *a subset* of these functions. Thus the computation can be recovered even if some of the nodes computing these functions slow down or fail. The challenge is to obtain low computation latency while using minimum extra resources. My solution in [4] uses erasure codes to construct such functions, and is $3\times-$ faster and consumes $2\times-$ fewer resources than task replication.

2. **Machine Learning (ML) based fault tolerance via drift mitigation** [5]: This work provides *indirect* fault tolerance through the use of ML for tasks like incident routing and resource allocation which, if performed correctly, can reduce the likelihood of encountering faults during computation. Conventional ML models lose accuracy in such settings as they are not equipped to deal with the changes (drifts) in data distribution caused by the time-varying nature of computing systems. The challenge is to design a ML solution that scales to production workloads and is accurate under data drift. My solution in [5], which selects the best out of a set of ML models based on similarity between training and test data, is $8\times-$ faster than prior work of comparable accuracy and is up to 20% more accurate than prior work of comparable speed.

Insights, Impact, and Outlook. The key takeaway from my work is the notion that statistical and machine learning tools can be effectively leveraged to model and predict the behaviour of computing systems. Representing phenomena like node failures and data drift in a statistical framework provides a language for communicating the underlying cause of the problem and a roadmap for designing an appropriate solution, along with a characterization of the accuracy, latency, cost, etc. of different approaches. In all my projects, the models are backed by evaluation on real systems. My work has received the **ACM SIGMETRICS 2020 Best Paper Award** and was also awarded the **Qualcomm Innovation Fellowship 2019**. My long-term goal in this space is to *design an end-to-end fault tolerant distributed computing framework leveraging both the algorithmic and machine learning approaches.* The strength of the ML solution is that it is agnostic to the nature of the computations and hence equally effective for all computations, while its weakness is that its effectiveness is not always guaranteed as deviation between training and test data used by the ML model can lead to incorrect predictions. The strength of the algorithmic solution is that its effectiveness is more predictable (adding more redundant functions leads to better fault tolerance), while its weakness is that it is currently limited to relatively simple computations for which the corresponding functions can be determined. Therefore a potential best-of-both-worlds approach can involve initial task assignment using the ML solution with redundancies added by the algorithmic approach acting as a buffer against errors in the ML approach.



(a) Coded Computing Framework



(b) Illustrating Near-Ideal Behaviour

Figure 1: (a) Worker i stores coded submatrix $\mathbf{A}_{\mathbf{e},i}$ and sends coded row-vector products to the master. Different workers may complete different number of row vector products (indicated by the extent of blue shading on the submatrices). The master aggregates all the coded row-vector products received and decodes them to recover $\mathbf{b} = \mathbf{Ax}$. (b) An ideal (dynamic recovery) computing scheme in this setting would move data from slow to fast workers. My coded schemes [4, 6, 7, 8] achieves near-ideal latency without data movement.

1 Algorithm Based Fault Tolerance Using Rateless Erasure Codes

The Core Problem: Straggling Nodes In Distributed Computing. Large-scale distributed computation is often bottlenecked by slow or unresponsive nodes *called stragglers* which cause the entire computation to be delayed [9]. The problem becomes worse with more nodes, which increases the odds of encountering stragglers. For eg. [9, Table 1] shows that the latency of executing many parallel tasks could be significantly larger (140 ms) than the median latency of a single task (1 ms).

Connection to Fault Tolerance. Node failure/unavailability can be viewed as an extreme case of node slowdown. In fact, delays incurred due to restarting nodes in the event of a disk crash, etc. can be reasons for *straggling*. Thus solutions for straggler mitigation can provide fault tolerance for the class of faults that can be considered as special cases of straggling.

My Solution. I first studied distributed computing in the context of distributed matrix-vector multiplication which forms the core of many important computing tasks like solving partial differential equations [10], forward and back propagation in neural networks [11], computing PageRank of graphs [12] etc. In this age of Big Data, most of these applications involve multiplying extremely large matrices and vectors and the computations cannot be performed efficiently on a single machine. Therefore algorithms like [13], [14] seek to speed-up matrix-vector multiplication by distributing the computation across multiple computing nodes. The individual nodes (the *workers*) perform their respective tasks in parallel while a central node (the *master*) aggregates the output of all these workers to complete the computation. This approach is naturally affected by faults and stragglers since the master needs to wait for *all* workers to complete their task.

My solution [4, 6] to the straggler problem is motivated by the following observations: a) linear combinations of matrix rows, when multiplied with the vector, will return linear combinations of the product elements, and b) the encoding functions of certain erasure codes - specifically *rateless fountain codes* [15, 16] - can be used to generate a nearly infinite stream of linear combinations of matrix rows such that a finite subset of these can be efficiently decoded to recover the original rows.

These observations can be used to speed up the distributed matrix-vector multiplication $\mathbf{b} = \mathbf{Ax}$ as follows (see fig. 1a). For a matrix with m rows, given by $\mathbf{A} = [\mathbf{a}_1^T, \ldots, \mathbf{a}_m^T]$, my strategy first generates (encodes) $m_e > m$ linear combinations of the rows and distributes them equally among the workers. Each of these linear combinations is generated by choosing d of the m rows uniformly at random and adding them. For example, if $d = 2$, and we choose rows $\mathbf{a}_1$ and $\mathbf{a}_3$, then the encoded row is $\mathbf{a}_1 + \mathbf{a}_3$. Each node multiplies its encoded rows with the input vector $\mathbf{x}$, and sends the results to the master. For e.g. the node with encoded row $(\mathbf{a}_1 + \mathbf{a}_3)$ would send $(\mathbf{a}_1 + \mathbf{a}_3)^T\mathbf{x} = b_1 + b_3$. The master collects these coded results and solves the associated system of equations to recover (decode) the original product $\mathbf{b} = \mathbf{Ax}$. The design of the distribution $\rho(d)$ [15] according to which d (number of rows in each linear combination) is chosen, enables successful decoding as soon as *any* $M' = m + o(m)$ encoded row-vector products are received. Moreover the encoding and decoding

steps are efficient with a complexity of $\mathcal{O}(m \log m)$ for both which enables seamless scaling to very large values of m. My theoretical analysis in [4] shows that, for large m, the latency of my scheme approaches that of dynamic recovery schemes like [17, 18, 19] without the overhead of task reallocation (see Figure 1b).

These benefits are illustrated in the experiments in [4] where I implemented my rateless coded matrix-vector multiplication scheme for a 11760×9216 matrix $\mathbf{A}$ with a sequence of vectors on a cluster of 70 Amazon EC2 nodes and obtained $3 \times -$ speedup and $2 \times -$ lower resource consumption than adding redundancy via replication. Similar gains were also seen for experiments on single-node parallel computing (using multiple processes) and serverless computing (on Amazon Lambda). I also showed in a follow-up work [7] that when the matrix is sparse (SpMV), and the problem of straggling is exacerbated by non-uniform density of matrix rows, my rateless coded approach combined with a simple load balancing strategy significantly outperforms the naive approach of assigning equal number of rows to each worker node.

Ongoing and Future Work. While my initial work focused on distributed linear computations represented by matrix-vector multiplication, a large number of real-world distributed computations such as gradients of neural network parameters [20], kernel functions between training and test points [21], and Taylor series approximations of expensive functions [22] are *non-linear*. Therefore my current focus is on designing algorithmic fault tolerance approaches for non-linear computations. The *motivating observation* in this line of work is that several expensive non-linear functions can be decomposed into sums of cheap functions. For e.g. batch gradients are sums of stochastic gradients, kernel functions are sums of pointwise kernels, and Taylor series is a sum of exponentials. Based on this, in [8] I designed a coded approach for distributed computation of $F(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x})$ (F is expensive, f_i's are cheap) which constructs random linear combinations of f_i's using rateless codes, computes them at individual workers, and reconstructs F from a *subset* of completed computations. Theoretical analysis and experiments on Amazon Lambda demonstrate the superiority of this approach over various uncoded baselines.

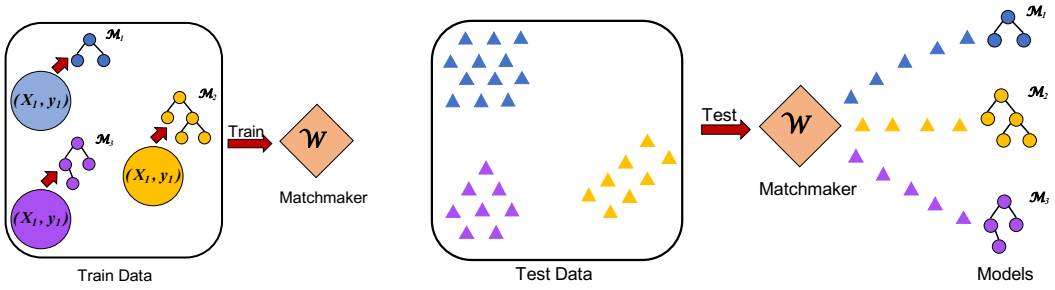
2 Fault Tolerance via Drift Mitigation in Machine Learning for Systems

The Core Problem: Data Drift in Machine Learning for Systems. Data from computing systems is inherently temporal and can change over time due to internal and external causes such as system upgrades, configuration changes, new workloads, and surging user demands [23]. Consequently, when ML is applied to systems applications like resource management [24], and incident routing [25], there can be a significant difference between the training and test data. Such difference is known as *data drift* and can cause large accuracy drops which has a significant negative impact on the corresponding application [24, 25, 26]. The data I obtained from ML models deployed in production systems in a large cloud provider shows that, despite frequent retraining, many of the models suffer from frequent and significant (up to 40%) accuracy drops.

Connection to Fault Tolerance. ML applications in cloud resource management [27] seek to maintain and improve the health of the underlying infrastructure through tasks like container scheduling (assigning containers to servers such that computation latency is minimized) [24], network incident routing (assigning issues to the right engineering team) [25], and compromise detection (identifying viruses and other security threats) [26]. Appropriate resource allocation, speedy incident resolution, and effective detection and elimination of threats, can all reduce the probability of node failure, thereby providing fault tolerance. Therefore mitigating issues like data drift in the associated ML models is crucial for effective fault tolerance.

My Solution. My system [5], called Matchmaker, addresses the two main types of data drift [28]: (1) *covariate shift* [29], where feature distribution changes over time but the underlying concept (the mapping from features to labels) stays the same; and (2) *concept drift* [30] where the underlying concept changes over time. To the best of my knowledge, Matchmaker is the first solution to the data drift problem that tackles both kinds of drift and is designed for ML deployments in large-scale systems.

The key idea underlying Matchmaker is to dynamically identify the batch of training data that has minimum drift with respect to *each* test sample, and use the ML model trained on that batch for inference (see Figure 2). To do this Matchmaker trains predictive models on each of the past T batches of training



(a) Training (Offline) (b) Testing (Online)

Figure 2: Overview of Matchmaker with 3 training batches. Predictive models ($\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$) are trained offline along with Matchmaker, $\mathcal{W}$ (Figure 2a). At test time (Figure 2b) $\mathcal{W}$ assigns each test point to the model from the most similar batch (same color) for prediction.

data (one model per batch). The accuracy of these models on the most recent batch of training data is used to rank them in terms of concept drift (higher accuracy indicates lower concept drift). Matchmaker then trains a random forest [31] that can *partition* the data space. This is used to rank training batches in terms of covariate shift (a batch with more points near a test point, under this partitioning, has lower covariate shift). The covariate shift and concept drift rankings are combined using a well known statistical aggregation method called the Borda Count [32] such that the highest ranked batch has minimum overall drift with respect to the training sample. The model trained on this batch is then used for prediction.

This idea has three major strengths. First, as Matchmaker selects the matching ML model for *each sample at test time*, it can adapt to changes in data drift within the same batch of test data, which often occurs in the real world. Second, Matchmaker is flexible to operator requirements as it provides operators with the means to select their own metrics for ranking which can easily be plugged in in place of the metrics we use in the paper. Third, Matchmaker is computationally efficient, with negligible overhead of computing the rankings which makes the inference latency comparable to methods using a single model for prediction. This is achieved by pre-computing the concept drift rankings and the traversal paths of the training data (used for the covariate shift rankings) and caching them in a hash-table which enables $\mathcal{O}(1)$ lookups at test time.

I evaluated Matchmaker on two production systems deployed in a commericial cloud provider for network incident routing (NETIR) and VM CPU utilization (VMCPU) respectively. The results show that, when compared to the currently deployed retraining mechanisms, Matchmaker (1) improves accuracy by up to 14% for NETIR, and up to 2% for VMCPU; and (2) reduces average operational costs by 18% for NETIR and 1% for VMCPU. Compared to a state-of-the-art concept drift solution, AUE [33], Matchmaker provides 8× and 4× faster ML predictions while achieving comparable accuracy. Compared to a recent solution DriftSurf [34], that runs faster than AUE, Matchmaker provides upto 20% and 6% improvement in accuracy with similar prediction speed. Matchmaker is currently being deployed in NETIR and a patent has been filed for the same.

Ongoing and Future Work. A key takeway from my research on different drift mitigation algorithms is that there is *no single algorithm* that works well for addressing all kinds of drift, at all times, for all datasets, and for all evaluation metrics. This is because every algorithm has its strengths and weaknesses with respect to drift mitigation, and also because different users care about different metrics; a user preferring high accuracy with no constraint on inference latency may prefer a slow but accurate approach like [33] while another user who requires low inference latency might sacrifice some accuracy for a faster method like [34]. Given this variabiity, I tested an approach which uses Multi-Armed Bandits [35] to search over the different drift mitigation algorithms (arms) to predict the best performing algorithm at every batch for a given dataset based on the history of its results for prior batches. While this is often able to predict the best algorithm correctly, I discovered a mismatch between the performance metrics in Multi-Armed Bandit algorithms, which are designed for average case scenarios and the preference of system designers who prefer to guard against the worst case scenario. Bridging this gap is the next step.

3 Other Projects

Outside of designing solutions for fault tolerance, I have worked on mitigating challenges that arise in real world deployments of ML algorithms.

Data-Efficient ML. The availability of huge labeled datasets [36] has played a key role in the success of machine learning, particularly deep learning in recent times. However, machine learning is increasingly being applied to areas like materials science [37], and disease detection [38] where it is challenging to procure large amounts of labeled data. In this line of work we proposed novel probabilistic ML algorithms [39, 40] which compute similarities between latent probabilistic representations of training and test data to make meaningful predictions that work well even in small-data settings due to the robustness of the probabilistic representations to overfitting. Experiments on a variety of datasets from the UCI repository for regression, *mini*-Imagenet and CUB datasets for few-shot classification, and a recently introduced COVID-19 chest X-Ray dataset for disease detection [38] demonstrate the superiority of our approaches in [39, 40] over conventional deep learning baselines.

Communication-Efficient ML. ML model training using a federated learning framework [41] is divided into communication rounds, where in each round, the central server has to estimate the mean of model updates or gradients sent by edge clients. When the vectors are high-dimensional, the communication cost of sending entire vectors may be prohibitive, and it may be imperative for them to use sparsification techniques. While most existing work on sparsified mean estimation is agnostic to the characteristics of the data vectors, in settings like federated learning, there may be spatial correlations (similarities in the vectors sent by different nodes) or temporal correlations (similarities in the data sent by a single node over different iterations of the algorithm) in the data vectors. In [42], we leverage these correlations by simply modifying the decoding method used by the server to estimate the mean. We provide an analysis of the resulting estimation error as well as experiments for PCA, K-Means and Logistic Regression, which show that our estimators consistently outperform more sophisticated and expensive sparsification methods.

References

[1] David A Rennels. Fault-tolerant computing. In *Encyclopedia of Computer Science*, pages 698–702. 2003.

[2] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[3] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[4] **Ankur Mallick**, Malhar Chaudhari, Utsav Sheth, Ganesh Palanikumar, and Gauri Joshi. Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3):1–40, 2019.

[5] **Ankur Mallick**, Kevin Hsieh, Behnaz Arzani, and Gauri Joshi. Matchmaker: Data drift mitigation in machine learning for large-scale systems. *Proceedings of Machine Learning and Systems*, 2022.

[6] **Ankur Mallick**, Malhar Chaudhari, and Gauri Joshi. Fast and efficient distributed matrix-vector multiplication using rateless fountain codes. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8192–8196. IEEE, 2019.

[7] **Ankur Mallick** and Gauri Joshi. Rateless codes for distributed computations with sparse compressed matrices. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2793–2797. IEEE, 2019.

[8] **Ankur Mallick**, Sophie Smith, and Gauri Joshi. Rateless codes for distributed non-linear computations. In *2021 11th International Symposium on Topics in Coding (ISTC)*, pages 1–5. IEEE.

[9] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[10] William F Ames. *Numerical Methods for Partial Differential Equations*. Academic Press, 2014.

[11] William Dally. High-performance hardware for machine learning. *NIPS Tutorial*, 2015.

[12] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[13] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*, volume 400. Benjamin/Cummings Redwood City, 1994.

[14] Geoffrey C. Fox, Steve W. Otto, and Anthony JG. Hey. Matrix algorithms on a hypercube i: Matrix multiplication. *Parallel Computing*, 4(1):17–31, 1987.

[15] Michael Luby. LT codes. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 271–271, 2002.

[16] Amin Shokrollahi. Raptor codes. *IEEE/ACM Transactions on Networking (TON)*, 14(SI):2551–2567, 2006.

[17] James Dinan, Stephen Olivier, Gerald Sabin, Jan Prins, P Sadayappan, and Chau-Wen Tseng. Dynamic load balancing of unbalanced computations using message passing. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8. IEEE, 2007.

[18] James Dinan, D Brian Larkins, Ponnuswamy Sadayappan, Sriram Krishnamoorthy, and Jarek Nieplocha. Scalable work stealing. In *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*, pages 1–11. IEEE, 2009.

[19] Aaron Harlap, Henggang Cui, Wei Dai, Jinliang Wei, Gregory R Ganger, Phillip B Gibbons, Garth A Gibson, and Eric P Xing. Addressing the straggler problem for iterative convergent parallel ml. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 98–111. ACM, 2016.

[20] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[21] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008.

[22] Alan Newell. *Nonlinear optics*. CRC Press, 2018.

[23] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. Machine learning: The high interest credit card of technical debt. In *SE4ML: Software Engineering for Machine Learning (NeurIPS Workshop)*, 2014.

[24] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Symposium on Operating Systems Principles (SOSP)*, 2017.

[25] Jiaqi Gao, Nofel Yaseen, Robert MacDavid, Felipe Vieira Frujeri, Vincent Liu, Ricardo Bianchini, Ramaswamy Aditya, Xiaohang Wang, Henry Lee, David Maltz, et al. Scouts: Improving the diagnosis process through domain-customized incident routing. In *Proceedings of the Annual conference of the Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM)*, 2020.

[26] Behnaz Arzani, Selim Ciraci, Stefan Saroiu, Alec Wolman, Jack W. Stokes, Geoff Outhred, and Lechao Diwu. PrivateEye: Scalable and privacy-preserving compromise detection in the cloud. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

[27] Ricardo Bianchini, Marcus Fontoura, Eli Cortez, Anand Bonde, Alexandre Muzio, Ana-Maria Constantin, Thomas Moscibroda, Gabriel Magalhaes, Girish Bablani, and Mark Russinovich. Toward ml-centric cloud platforms. *Communications of the ACM*, 63(2):50–59, 2020.

[28] Jose G Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V Chawla, and Francisco Herrera. A unifying view on dataset shift in classification. *Pattern recognition*, 45(1):521–530, 2012.

[29] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2), 2000.

[30] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1), 1996.

[31] Alex Davies and Zoubin Ghahramani. The random forest kernel and other kernels for big data from random partitions. *arXiv preprint arXiv:1402.4293*, 2014.

[32] Nihar B Shah and Martin J Wainwright. Simple, robust and optimal ranking from pairwise comparisons. *The Journal of Machine Learning Research*, 18(1), 2017.

[33] Dariusz Brzezinski and Jerzy Stefanowski. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1), 2013.

[34] Ashraf Tahmasbi, Ellango Jothimurugesan, Srikanta Tirthapura, and Phillip B Gibbons. Driftsurf: A risk-competitive learning algorithm under concept drift. *arXiv preprint arXiv:2003.06508*, 2020.

[35] Aleksandrs Slivkins. Introduction to multi-armed bandits. *arXiv preprint arXiv:1904.07272*, 2019.

[36] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[37] Jize Zhang, Bhavya Kailkhura, and T Han. Leveraging uncertainty from deep learning for trustworthy materials discovery workflows. *arXiv preprint arXiv:2012.01478*, 2020.

[38] Joseph Paul Cohen, Paul Morrison, and Lan Dao. Covid-19 image data collection. *arXiv preprint arXiv:2003.11597*, 2020.

[39] **Ankur Mallick**, Chaitanya Dwivedi, Bhavya Kailkhura, Gauri Joshi, and T Yong-Jin Han. Deep kernels with probabilistic embeddings for small-data learning. In *Uncertainty in Artificial Intelligence*, pages 918–928. PMLR, 2021.

[40] **Ankur Mallick**, Chaitanya Dwivedi, Bhavya Kailkhura, Gauri Joshi, and T Han. Probabilistic neighbourhood component analysis: Sample efficient uncertainty estimation in deep learning. *arXiv preprint arXiv:2007.10800*, 2020.

[41] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[42] Divyansh Jhunjhunwala, **Ankur Mallick**, Advait Gadhikar, Swanand Kadhe, and Gauri Joshi. Leveraging spatial and temporal correlations in sparsified mean estimation. *Advances in Neural Information Processing Systems*, 34, 2021.