My goal is to build **programming languages, compilers, and runtime systems** for efficient and secure **AI systems** that make it easy to develop and deploy end-to-end AI pipelines on large-scale *distributed* clusters, while utilizing *heterogeneous* architectures.

Distributed and heterogeneous hardware are increasingly used to meet the performance requirements of today's applications. These architectures are tedious to program because of their heterogeneity, as devices such as CPU, GPU, FPGA, and TPU provide different programming abstractions and may have disjoint memories, even if they are on the same machine. Moreover, applications are written by experts in the application domain rather than experts in parallel programming. Consequently, it is hard for programmers to: (1) write efficient code for each device, (2) distribute execution across devices, (3) orchestrate communication between devices, and (4) port code to new emerging devices. To address these problems, my goal is to design programming languages that provide a common domain-specific interface for all devices, and build compilers and runtime systems that generate efficient architecture-specific code for each device, distribute computation among devices, and move data between them efficiently.

The datasets and models used by AI applications are growing in size. With this growth, there is an increasing need to exploit sparsity in the dataset or the model for efficiency. For example, most of the weights (model) learned in many typical *deep neural networks* (DNN) may be pruned for faster inference without loss in accuracy. On the other hand, privacy of the datasets used must be preserved in many applications, especially if they run on public clouds. For example, applications may need to run DNN inference on medical or financial data without violating the data owner's privacy. Fully-Homomorphic Encryption (FHE) enables computation on encrypted data without requiring the secret key. In my view, an FHE scheme is akin to yet another heterogeneous device, so compilers can generate code for them to enforce privacy.

My dissertation research focuses on **distributed and heterogeneous graph analytics** and **privacy-preserving neural network inferencing**, and introduces new techniques in systems for sparse computing and privacy-preserving applications respectively. I designed and built a graph analytics system [6, 7] that partitions graphs [17] and optimizes communication on distributed, heterogeneous architectures, while providing application-specific fault-tolerance [8]. This system was an order of magnitude faster than existing systems at scale. I designed a new language [9] for FHE and built an optimizing compiler [12] that translates DNN inference to run on encrypted data using FHE efficiently, while guaranteeing security and accuracy. The generated codes were an order of magnitude faster than expert-written codes.

During my time at the graph AI startup, Katana Graph, I interacted with customers to understand their challenges in using AI systems. To address their major concerns, I led the graph engine team to build a distributed cloud platform for computing AI, analytics, and queries efficiently on large-scale graphs. My team built an in-memory and on-storage log-structured representation for labelled property graphs that is compact for sparse node and edge properties, while being efficient for both reading and updating the graph topology and properties. I designed and built the distributed graph querying engine that minimizes latency of business intelligence queries and scales well on distributed hosts. Katana Graph was significantly faster than our competition in the end-to-end time for a typical graph AI pipeline.

Overall, I have collaborated with researchers in different areas, including programming languages, systems, cryptography, security, and theory, and worked across the software stack such as algorithms, compilers, and low-level runtimes. The common theme in my research is to find important application domains, design programming abstractions for each application domain, and exploit domain knowledge at the right layer of the software stack to determine the right trade-off between productivity, portability, performance, and privacy. My vision is to build easy-to-use AI systems that enables application developers to reason about this trade-off.

Research Contributions

Distributed and Heterogeneous Graph Processing: Graph analytics systems provide a simple programming model to develop applications that analyze graphs like ranking web pages in search engines, finding clusters in biological networks, evaluating recommender systems, and finding shortest routes in maps. Such systems must handle large graphs like the Facebook friends graph, which has more than 2 billion nodes and 400 billion edges. Shared-memory systems like Galois [23], Ligra [27], and IrGL [24] are efficient but have limited memory and compute resources. Prior distributed-memory systems either did not scale or were not competitive with shared-memory systems. Moreover, they were restricted to CPUs and there was no way to reuse their techniques to leverage accelerators like GPUs. My dissertation addressed these issues.

Programming Model and Runtime: In my PLDI 2018 paper [6] and PACT 2019 paper [7] (one of the four Best Paper Nominees), I introduced a novel approach to build distributed graph analytics systems that exploits heterogeneity in processor types and partitioning policies. Programmers write applications in Galois for CPU or IrGL for GPU, and interface with **Gluon**, a communication-optimizing substrate. Gluon partitions the graph and offloads each partition to a CPU or GPU. I designed a new asynchronous execution model called **Bulk-Asynchronous Parallel (BASP)**. Exploiting the domain knowledge that graph analytics applications are resilient to stale reads, I introduced a novel way to synchronize the partitions *eventually*. I also created **communication optimizations** that exploit structural and temporal invariants of graph partitioning policies. Existing graph analytics (or sparse matrix) systems for a single CPU or GPU can use Gluon to scale out to distributed clusters with little effort.

Gluon's modularity and abstraction does not come at the cost of performance. Execution time of Gluon improved by ∼3× on average due to its communication optimizations, enabling it to scale well up to 256 CPUs and 64 GPUs. Gluon was faster than the prior state-of-the-art distributed CPU-only system, Gemini [29], by ∼4× and ∼5× on average using CPUs and GPUs respectively. It was on average ∼12× faster than the only other distributed GPU-only system, Lux [22], at scale. Gluon is the only asynchronous distributed GPU graph system currently.

Resilience: My ASPLOS 2019 paper [8] tackles *fail-stop* faults (or machine crashes). The traditional way to tolerate faults is to *checkpoint* the application's state periodically and *roll-back* the state to the last checkpoint when a fault occurs. Faults are rare, but checkpointing has overheads, even when no faults occur. My insight was that to recover from faults, it is sufficient to restart the computation (or *roll-forward*) from a state that will ultimately produce the correct result. Such states are called *valid* states. I classified graph algorithms and designed class-specific recovery protocols called **Phoenix** to compute a valid state from the current state. The computation function can be provided by programmers with little effort. Phoenix was incorporated into Gluon to make it **resilient to fail-stop faults**. Phoenix not only has no overhead in the absence of faults, but also outperforms checkpointing when few faults occur.

Software Stack: My work on Gluon has led to research on different aspects of graph processing. I collaborated with researchers to: (1) build a distributed training framework [16], on top of Gluon, for a class of applications like Word2Vec that use Skip-gram-like models to generate embeddings; (2) optimize the graph analytics runtime for byte-addressable memory [14]; (3) build the **Abelian** compiler [13] that compiles code written in its domain-specific language to run on distributed and heterogeneous clusters by generating CUDA code as well as the required communication code using Gluon; (4) develop an alternative to the Message Passing Interface (MPI) for graph analytics called **Lightweight Communication Interface (LCI)** [5]; (5) study the impact of different partitioning policies on execution time [15, 21] and build a fast **Customizable, Streaming Partitioner (CuSP)** [17]; and (6) develop distributed imple-

mentations of efficient algorithms for betweenness centrality [19], triangle counting [18], and belief propagation [28]; and (7) build programming frameworks [2, 1] for graph pattern mining (GPM) problems such as motif counting. During this time, I mentored graduate students and post-doctoral scholars, namely Loc Hoang, Bozhi You, Hochan Lee, Vishwesh Jatala, and Xuhao Chen. These works were published in VLDB, IPDPS, PPoPP, ICS, and Euro-Par.

Privacy-Preserving Deep Neural Network (DNN) Inference: Fully-Homomorphic Encryption (FHE) enables offloading both storage and computation of sensitive data to public clouds, without trusting software vendors, hardware vendors, or any third party with their secret key. However, developing FHE applications requires cryptographic expertise. In my PLDI 2019 paper [12], I built an end-to-end software stack called **CHET** for compiling tensor programs like DNN inference to run on FHE libraries [26, 20] that support *fixed-point* arithmetic. In my PLDI 2020 paper [9], I designed a new encrypted vector arithmetic language and compiler called **EVA** for developing general-purpose FHE applications.

Runtime: FHE schemes allow batching thousands of *plaintext* elements into a *ciphertext* vector and perform element-wise vector operations to amortize the cost. Like Intel MKL libraries, which have different implementations of linear algebra operations, I built a **library of homomorphic tensor operations** with different ways to map or *layout* tensors onto vectors.

Compiler: Encryption parameters influence the FHE computation. Setting these parameters low can make the computation insecure, whereas setting them large can increase the cost of homomorphic operations. Moreover, when these parameters are not sufficiently large, the encrypted result becomes corrupted. For two state-of-the-art FHE schemes [4, 3], I introduced compiler analysis to determine the minimum required encryption parameters for a program that ensures it is correct and secure. I then created compiler analysis to estimate the cost of a program using a cost model for the two schemes. CHET explores different layouts, **determines encryption parameters** and cost for each layout, and picks the best-performing one.

EVA is designed to be an intermediate representation that is a backend for other domain-specific compilers. I modified CHET to generate EVA programs for DNN inference. I built an optimizing compiler for EVA that generates correct and secure programs, while hiding all the complexities of the target FHE scheme. The compiler eliminates all common runtime exceptions and optimally inserts FHE instructions like rescaling and modulus switching.

CHET was the first compiler for DNN inference using FHE. It enabled homomorphic inference of deeper DNNs than was viable by experts' programming. It also allowed using FHE schemes that are much harder to program. Due to this and other optimizations, CHET was an order of magnitude faster than expert-written codes, even for small DNNs. CHET, when retargeted onto EVA, is *only* 2-3 orders of magnitude slower than simple, unencrypted inference — a new landmark for FHE. EVA also enabled a wider adoption of FHE for other applications.

Research Agenda

Hardware specialization is on the rise with companies building custom silicon for AI. The models are growing exponential in size; billions of weights are now common. The datasets to train these models are also growing larger. I believe programming languages, compilers, and runtime systems are essential in helping AI scientists and users navigate this space without sacrificing performance or privacy and I am excited to pursue research in that direction.

Feature Generation: Consider the elements in a dataset such as unique words in a text corpus, protein molecules in a biological dataset, and accounts in a financial dataset. Both training and inference rely on features already generated for the elements. The quality of

the features directly impact the accuracy of the trained model. One way to generate better features is to exploit any structure in the dataset like the interactions between proteins or the transactions between accounts. To do so, the dataset can be abstracted as a graph, where the elements form the nodes and their interactions form the edges, and graph analytics and graph queries can be used to generate more contextualized features for the elements. I plan to build systems that make it easier to compose such better feature generators.

Large-Scale Training: Large datasets and models take a long time to train. Reducing the time or cost for training can have a huge impact, especially in enabling more hyper-parameter tuning. The key to this is to better utilize distributed and heterogeneous hardware. I plan to leverage my experience with polyhedral compiler techniques [11, 10, 25], to generate locality optimized code for heterogeneous architectures. Building on my experience with Gluon, I want to build distributed schedulers and communication systems that are optimized for training.

Lightweight Inference: AI users may need to perform inference using large models on public clouds or IoT devices. To preserve the privacy of the datasets and/or the models on public clouds, I intend to extend my work on CHET and EVA to support running more advanced and bigger AI models using FHE. In addition, IoT devices are resource constrained compared to the hardware used for training. I intend to build systems that can transparently prune the learnt weights and reduce the model size without significant loss in accuracy.

References

[1] Xuhao Chen, Roshan Dathathri, Gurbinder Gill, Loc Hoang, and Keshav Pingali. Sandslash: A Two-Level Framework for Efficient Graph Pattern Mining. In *Proceedings of the ACM International Conference on Supercomputing*, ICS '21, page 378–391, New York, NY, USA, 2021. Association for Computing Machinery.

[2] Xuhao Chen, Roshan Dathathri, Gurbinder Gill, and Keshav Pingali. Pangolin: An Efficient and Flexible Graph Mining System on CPU and GPU. *Proc. VLDB Endow.*, 13(8):1190–1205, apr 2020.

[3] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. pages 347–368, 2019.

[4] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. pages 409–437, 2017.

[5] Hoang-Vu Dang, Roshan Dathathri, Gurbinder Gill, Alex Brooks, Nikoli Dryden, Andrew Lenharth, Loc Hoang, Keshav Pingali, and Marc Snir. A Lightweight Communication Runtime for Distributed Graph Analytics. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2018.

[6] Roshan Dathathri, Gurbinder Gill, Loc Hoang, Hoang-Vu Dang, Alex Brooks, Nikoli Dryden, Marc Snir, and Keshav Pingali. Gluon: A Communication-Optimizing Substrate for Distributed Heterogeneous Graph Analytics. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '18, New York, NY, USA, 2018. ACM.

[7] Roshan Dathathri, Gurbinder Gill, Loc Hoang, Hoang-Vu Dang, Vishwesh Jatala, V. Krishna Nandivada, Marc Snir, and Keshav Pingali. Gluon-Async: A Bulk-Asynchronous System for Distributed and Heterogeneous Graph Analytics. In *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 15–28, Sep. 2019.

[8] Roshan Dathathri, Gurbinder Gill, Loc Hoang, and Keshav Pingali. Phoenix: A Substrate for Resilient Distributed Graph Analytics. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, pages 615–630, New York, NY, USA, 2019. ACM.

[9] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madan Musuvathi. EVA: An Encrypted Vector Arithmetic Language and Compiler for Efficient Homomorphic Computation. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, page 546–561, New York, NY, USA, 2020. Association for Computing Machinery.

[10] Roshan Dathathri, Ravi Teja Mullapudi, and Uday Bondhugula. Compiling Affine Loop Nests for a Dynamic Scheduling Runtime on Shared and Distributed Memory. *ACM Trans. Parallel Comput.*, 3(2):12:1–12:28, July 2016.

[11] Roshan Dathathri, Chandan Reddy, Thejas Ramashekar, and Uday Bondhugula. Generating Efficient Data Movement Code for Heterogeneous Architectures with Distributed-Memory. In *Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques*, PACT '13, pages 375–386, Piscataway, NJ, USA, 2013. IEEE Press.

[12] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. CHET: An Optimizing Compiler for Fully-Homomorphic Neural-Network Inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pages 142–156, New York, NY, USA, 2019. ACM.

[13] Gurbinder Gill, Roshan Dathathri, Loc Hoang, Andrew Lenharth, and Keshav Pingali. Abelian: A Compiler for Graph Analytics on Distributed, Heterogeneous Platforms. In Marco Aldinucci, Luca Padovani, and Massimo Torquati, editors, *Euro-Par 2018: Parallel Processing*, pages 249–264, Cham, 2018. Springer International Publishing.

[14] Gurbinder Gill, Roshan Dathathri, Loc Hoang, Ramesh Peri, and Keshav Pingali. Single Machine Graph Analytics on Massive Datasets Using Intel Optane DC Persistent Memory. *Proc. VLDB Endow.*, 13(8):1304–1318, apr 2020.

[15] Gurbinder Gill, Roshan Dathathri, Loc Hoang, and Keshav Pingali. A Study of Partitioning Policies for Graph Analytics on Large-scale Distributed Platforms. volume 12 of *PVLDB*, 2018.

[16] Gurbinder Gill, Roshan Dathathri, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, and Olli Saarikivi. Distributed Training of Embeddings using Graph Analytics. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 973–983, 2021.

[17] Loc Hoang, Roshan Dathathri, Gurbinder Gill, and Keshav Pingali. CuSP: A Customizable Streaming Edge Partitioner for Distributed Graph Analytics. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2019.

[18] Loc Hoang, Vishwesh Jatala, Xuhao Chen, Udit Agarwal, Roshan Dathathri, Gurbinder Gill, and Keshav Pingali. DistTC: High Performance Distributed Triangle Counting. In *Proceedings of the IEEE International Conference on High Performance Extreme Computing*, HPEC'19, 2019.

[19] Loc Hoang, Matteo Pontecorvi, Roshan Dathathri, Gurbinder Gill, Bozhi You, Keshav Pingali, and Vijaya Ramachandran. A Round-Efficient Distributed Betweenness Centrality Algorithm. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, PPoPP '19, pages 272–286, New York, NY, USA, 2019. ACM.

[20] Cryptography Lab in Seoul National University. Homomorphic encryption for arithmetic of approximate numbers (heaan). `https://github.com/snucrypto/HEAAN`.

[21] Vishwesh Jatala, Roshan Dathathri, Gurbinder Gill, Loc Hoang, V. Krishna Nandivada, and Keshav Pingali. A Study of Graph Analytics for Massive Datasets on Distributed Multi-GPUs. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 84–94, 2020.

[22] Zhihao Jia, Yongkee Kwon, Galen Shipman, Pat McCormick, Mattan Erez, and Alex Aiken. A distributed multi-gpu system for fast graph processing. *Proc. VLDB Endow.*, 11(3):297–310, November 2017.

[23] Donald Nguyen, Andrew Lenharth, and Keshav Pingali. A Lightweight Infrastructure for Graph Analytics. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 456–471, New York, NY, USA, 2013. ACM.

[24] Sreepathi Pai and Keshav Pingali. A compiler for throughput optimization of graph algorithms on gpus. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2016, pages 1–19, New York, NY, USA, 2016. ACM.

[25] Mahesh Ravishankar, Roshan Dathathri, Venmugil Elango, Louis-Noël Pouchet, J. Ramanujam, Atanas Rountev, and P. Sadayappan. Distributed Memory Code Generation for Mixed Irregular/Regular Computations. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP 2015, pages 65–75, New York, NY, USA, 2015. ACM.

[26] Microsoft SEAL (release 3.3). `https://github.com/Microsoft/SEAL`, June 2019. Microsoft Research, Redmond, WA.

[27] Julian Shun and Guy E. Blelloch. Ligra: a lightweight graph processing framework for shared memory. In *Proceedings ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, PPoPP '13, pages 135–146, 2013.

[28] Alexandra Ulanov, Manish Marwah, Mijung Kim, Roshan Dathathri, Carlos Zubieta, and Jun Li. Sandpiper: Scaling Probabilistic Inferencing to Large Scale Graphical Models. In *2017 IEEE International Conference on Big Data (Big Data)*, Dec 2017.

[29] Xiaowei Zhu, Wenguang Chen, Weimin Zheng, and Xiaosong Ma. Gemini: A Computation-centric Distributed Graph Processing System. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 301–316, Berkeley, CA, USA, 2016. USENIX Association.