# PYXIS: AN OPEN-SOURCE PERFORMANCE DATASET OF SPARSE ACCELERATORS

*Linghao Song, Yuze Chi, and Jason Cong*

University of California, Los Angeles

{linghaosong,chiyuze,cong}@cs.ucla.edu

## ABSTRACT

Customized accelerators provide gains of performance and efficiency in specific domains of applications. Sparse data structures and/or representations exist in a wide range of applications. However, it is challenging to design accelerators for sparse applications because no architecture or performance-level analytic models are able to fully capture the spectrum of the sparse data. Accelerator researchers rely on real execution to get precise feedback for their designs. In this work, we present PYXIS, a performance dataset for customized accelerators on sparse data. PYXIS collects accelerator designs and real execution performance statistics. Currently, there are 73.8 K instances in PYXIS. PYXIS is open-source, and we are constantly growing PYXIS with new accelerator designs and performance statistics. PYXIS can be a benefit to researchers in the fields of accelerator, architecture, performance, algorithm and many related topics.

***Index Terms*—** Dataset, Sparse Accelerator, Performance, Customized Architecture.

## 1. INTRODUCTION

Conventional processors (CPUs) are gradually becoming inefficient for processing of big-data applications because of the diminishing gain of Moore's Law [1]. Domain specific accelerators [2, 3, 4, 5] benefiting from the customization in architectures, hardware components, and even with the integration of the customization in compilers and the whole computing stack, have been explored for higher system performance and energy efficiency in many application domains such as deep learning [6, 7, 8, 9, 10, 11]

Because a few parameters can determine the whole execution of a dense workload, it is relatively easy to use an analytic method to model the performance of accelerators for dense workload and thus guide the design of dense accelerators. For example, we can use as few as six parameters to describe the processing of a convolutional layer. Thus, Zhang et al. [8] build a roofline [12] based model for the design space exploration of deep learning accelerators, and Wang et al. [6] employ a polyhedral model for automatic systolic array compilation for dense applications.

This dataset is publicly available at https://github.com/linghaosong/Pyxis.

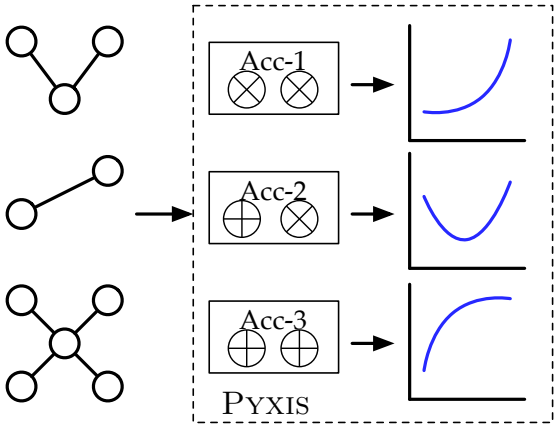


**Fig. 1**. PYXIS collects the performance statistics and the corresponding accelerator designs.

Sparse structures such as sparse matrices and graphs encode the properties and connections, representing data from nature, society, and a general perspective. Besides the computation-related optimizations, accelerator design [13, 14, 15, 16, 17, 18] for sparse workloads faces more challenges such as memory optimization, communication reduction, and efficient data format. However, it is impossible to use an analytic model to guide the design of accelerators for sparse applications because we need the whole sparse structure to determine the processing, and a sparse structure is usually huge. Therefore, instead of analytic models, real execution statistics are helpful for the guidance of accelerator design.

There lacks a collection of performance statistics of accelerators on sparse workloads. However, acquiring enough real execution statistics is very difficult. Before the real execution, an accelerator needs to be either implemented as an application-specific integrated circuit (ASIC) chip or prototyped as a field-programmable gate array (FPGA) based accelerator. The ASIC implementation flow may take several months, and the FPGA prototype flow requires many hours to several days. The time-consuming accelerator implementation flows have a deleterious impact on the generation of massive volumes of real execution statistics. On top of this, most accelerators can only support a fixed-size application and are not general-purpose, which means one accelerator design usually results in one or only a few performance statistics.

The whole flow of running a sparse workload consists of at least four components. They are sparse data input, an application, an accelerator, and performance statistics. We present PYXIS, an open-source performance dataset for customized accelerators on sparse data. We focus on the performance data and the accelerator design in PYXIS, as shown in Fig. 1 To date, we have collected 73.8 K performance instances. We use 2,637 out of all 2,893 sparse matrices in SuiteSparse [19] to evaluate sparse accelerators. The matrices not in use are out of memory. We run sparse-matrix dense-matrix multiplication on two FPGA platforms (Xilinx Alveo U250 and U280) and two GPU platforms (Nvidia Tesla K80 and V100) to collect the real execution statistics, i.e., the latency and throughput. We will keep growing PYXIS by adding the latest accelerator designs and performance statistics. We will also issue regular open calls to the whole community to enrich the collections in PYXIS.

## 2. RELATED WORKS

There are a few benchmarks for the evaluation of computing systems and customized accelerators. SPEC CPU Benchmark [20] is the most widely used benchmark suite for evaluating CPU performance and CPU architecture. Rodinia [21] is a benchmark suite for multi-core CPUs and GPUs. MLPerf [22] provides a collection of machine learning applications for system evaluation. For the evaluation of high-level synthesis (HLS) based FPGA accelerators, Rodinia-hls [23] provides the HLS versions of Rodinia applications, and Rosetta [24] includes the latest deep learning applications. For sparse data collections, SuiteSparse [19] is the widely used collection that contains more than two thousand sparse matrices (graphs) from a wide range of application domains. The above-mentioned benchmarks/collections mainly focus on the applications and sparse structures. However, there is a lack of a performance collection of customized accelerators for sparse workloads.

## 3. PYXIS DATASET

### 3.1. Method

**Sparse input data and applications.** PYXIS collects sparse accelerators and performance statistics. We use 2,637 out of all 2,893 sparse matrices in SuiteSparse [19] as the sparse input data. The sparse application we use is sparse-matrix dense-matrix multiplication (SpMM). SpMM computes $\mathbf{C} = \alpha\mathbf{A} \times \mathbf{B} + \beta\mathbf{C}$. where $\mathbf{A}$ is a sparse matrix, $\mathbf{B}$ and $\mathbf{C}$ are dense matrices, and $\alpha$ and $\beta$ are two constant scalars. We select SpMM because it is a widely used sparse application, and the configuration of one SpMM $(M, K, N)$ is flexible. An input sparse matrix/graph determines the $M$ and $K$, but users select the $N$ value for their applications. So we can change the $N$ value from 8 to 512 to generate seven different application settings for each sparse input.

**Table 1**. Specifications of four platforms.

| | Technology | Frequency | Bandwidth |
|---|---|---|---|
| **Alveo U250** | 16 nm | 190 MHz | 77 GB/s |
| **Alveo U280** | 16 nm | 197 MHz | 460 GB/s |
| **Tesla K80** | 28 nm | 562 MHz | 480 GB/s |
| **Tesla V100** | 12 nm | 1297 MHz | 900 GB/s |

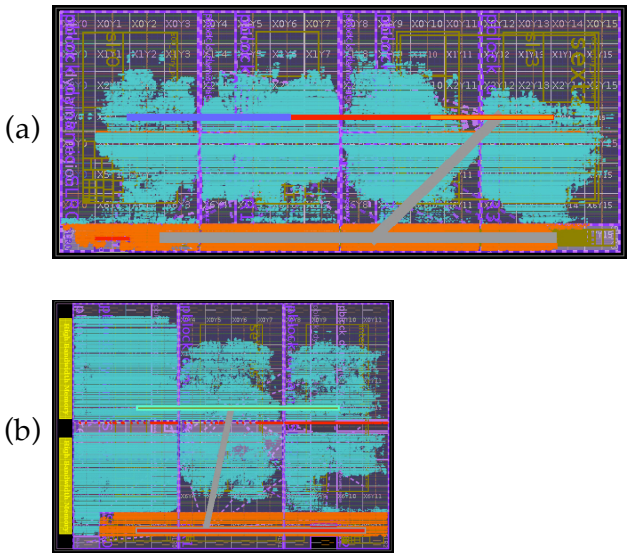


**Fig. 2**. Accelerator layouts on (a) Alveo U250 FPGA and (b) Alveo U280 FPGA.

**Sparse accelerators.** As previously discussed, one accelerator design is usually for a fixed-size application. To support a different application or application size, we need to run the time-consuming accelerator prototype/manufacture flow. Thanks to recent advances [25, 26] in accelerator design, Sextans [26] and GraphLily [25] support an arbitrary SpMM with only one hardware prototype generated. We use the Sextans architecture in this work. The accelerators are described in HLS C/C++ code. To generate the accelerators, we use the high level synthesis tool Vitis 2020.2.

We use two Xilinx FPGA accelerator cards, Alveo U250 and Alveo U280, for the accelerators. The two FPGA cards have different memory bandwidths and different amounts of logic resources. The Alveo U250 FPGA is equipped with a 77 GB/s DRAM, while the Alveo U280 FPGA is equipped with a 460 GB/s high bandwidth memory (HBM). There are four super logic regions (SLRs) on an Alveo U250 but three SLRs on an Alveo U280. We use the two platforms to evaluate accelerator designs with different hardware constraints. Fig. 2 shows the layouts of the generated hardware. Besides bandwidth, the frequency is another factor affecting the performance of accelerators. The layout (placing and routing) affects the frequency. Listed in Table 1, Alveo U250 is 190 MHz, whereas the frequency of Alveo U280 is 197 MHz. Readers who are

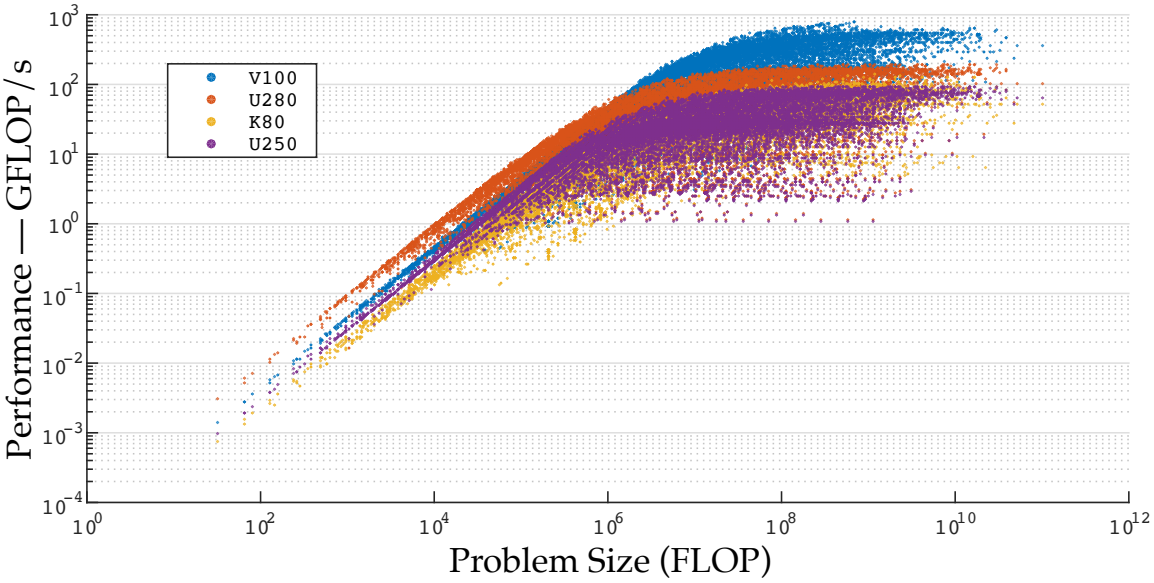


**Fig. 3**. Throughputs (in GFLOP/s) v.s. problem size (in FLOP).

interested in the accelerator architecture can find more details in [26].

We include the performance on GPUs because the GPU is another type of accelerator architecture, i.e., single instruction multiple thread (SIMT) architecture. We use two GPUs, Tesla K80 and Tesla V100. In addition, we employ CuSPARSE routine `csrmm` to run SpMMs with CUDA 10.2. Table 1 illustrates the frequency and memory bandwidth of the two GPU platforms. We also include performance statistics from an Intel Xeon Gold 6244 Processor with 16 threads and 180 GB memory.

### 3.2. Performance Data

For each SpMM run, we collect two performance statistics: (1) the latency in millisecond (ms) and (2) the processing throughput in giga floating-point operations per second (GFLOP/s). One SpMM run generates one data sample. In total, we collected 73.8 K instances in PYXIS. Each data sample contains the two performance statistics and the specification of the sparse matrix and the SpMM.

Fig. 3 illustrates the performance scatters of the four platforms. We define the problem size as the number of floating point operations for each SpMM. The performance ranges from $10^{-3}$ GFLOP/s to $10^{3}$ GFLOP/s. The problem size spans a vast range from $10^{1}$ to $10^{11}$ FLOP. We see that PYXIS contains rich and wide data distribution for both performance statistics and the problem size. The rich and wide distribution can provide sufficient information for the following researchers who use PYXIS dataset. For example, users who run an SpMM with a problem size smaller than $10^{6}$ FLOP will definitely select the U280 FPGA because of higher performance. For hardware accelerator researchers, it is worthwhile to explore why SIMT architectures get higher performance than FPGAs on a large-size problem.

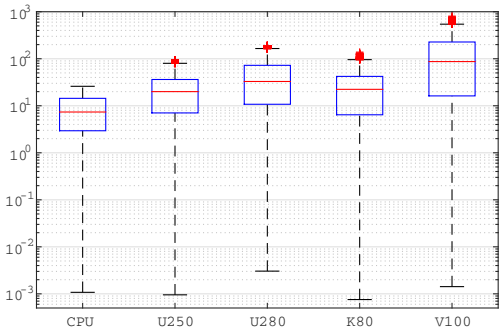


**Fig. 4**. Performance distribution (GFLOP/s).

Fig. 4 presents the box plots of performance on four platforms. The peak throughput of the CPU, Alveo U250, U280, Tesla K80, V100 is 25.95 GFLOP/s, 94 GFLOP/s, 191 GFLOP/s, 139 GFLOP/s, 800 GFLOP/s, respectively. The standard deviation of performance of the five platforms is 6.74, 22.4, 45.0, 23.4, 152.1, respectively. For each platform, the performance also covers a wide range. Thus, the performance information can help architecture researchers explore the behaviors of the different architectures on various sparse workloads.

## 4. POTENTIAL APPLICATIONS

We discuss three potential applications which can benefit from the PYXIS dataset.

**Accelerator design.** Although customized architectures can improve the performance and efficiency for the processing of

specific applications, it is nontrivial to design an accelerator. The systolic array architectures are suitable for accelerating dense tensor multiplication, but there are many versions of systolic array architectures. The first generation of Google's TPU [11] employs an output reuse dataflow for the systolic array. In the academic systolic array architecture Eyeriss [10], Chen et al. identify three existing data flows, i.e., weight stationary, output stationary, and no local reuse, and present a new data flow called row stationary. These data flow designs are all empirical. Later, Cong and Wang [6, 7] use a polyhedral model to automatically design the dataflows according to the size of the dense tensor multiplication. We see that the research for an optimized systolic array requires the effort from many researchers for many years. It is even more difficult for accelerator design for sparse workloads. We hope PYXIS can provide insights, such as bandwidth and processing unit utilization, for accelerator researchers. Besides manual architectural optimizations, automatic design space exploration [27, 28] and a general automatic architecture search are significantly efficient for accelerator design. The automatic search employs AI-based methods. For example, [28, 29] use GNNs for optimization. AI based searching relies heavily on training samples to achieve a better result. The performance data in PYXIS is helpful for the training of searching models in automatic accelerator design.

**Performance prediction on a new accelerator.** Evaluating a new accelerator takes a long time where the bottleneck is the prototype or the manufacture flows. A fast and accurate prediction will save time for accelerator researchers. The accelerator collections in PYXIS are essential for performance prediction. Currently, we do not see a model which can take an accelerator design as input and predict the performance results fast and accurately. One of the challenges is how to encode the accelerator design. However, besides GNNs, the advance in natural language processing (NLP) models [30] can be helpful. We describe accelerators in HLS C/C++ codes. NLP models can be powerful tool to encode the code files of accelerators. We believe an NLP-based performance mode can provide fast and accurate feedback by learning from numerous accelerators. However, the performance model requires further efforts from accelerator researchers and NLP/algorithm researchers.

**Task offloading and choice of computing platforms.** Modern computing systems consist of various processing units, including CPUs, GPUs, and accelerators. It is a challenge for users to efficiently offload the computing tasks to the available processing units. The performance data in PYXIS can guide the choice of the computing platform for a specific task. For example, if a user runs multiple SpMMs on a cluster equipped with a V100 GPU and a U280 FPGA and the user plans to run an SpMM where the problem size is $10^4$ FLOP, she is likely to run it on the FPGA suggested by the performance data in Fig. 3. However, if the problem size is $10^{10}$ FLOP, she will likely run it on the V100 GPU.

**Large-size graph classification/regression.** Besides the algorithm researchers focusing on the interdisciplinary topics of accelerator and algorithm, PYXIS can also benefit pure machine learning application researchers. PYXIS provides meaningful labels for many sparse matrices and graphs. Graph classification/regression [31, 32] focus on the classification/regression on the graph level. Current graph classification/regression works on small-size graphs where the node number is a few hundred. One challenge that prevents graph classification/regression from working on large-size graphs is the lack of labels. Although existing datasets such as SuiteSparse [19] contain many large-size graphs where the node number of a graph is up to one million, the labels for graphs are only ID number or datatype but not meaningful properties. PYXIS labels the large-size graphs with performance metrics. We believe that PYXIS can help the algorithm advance in the large-size graph classification/regression.

## 5. CONCLUSION AND FUTURE WORK

We present PYXIS, a performance dataset for customized accelerators running on sparse data. PYXIS collects accelerator designs described in HLS C/C++ codes and real execution performance statistics (latency and throughput). Currently, there are 73.8 K instances in PYXIS. PYXIS can benefit researchers in the fields of accelerator, architecture, performance, algorithm, and other topics for potential applications, including accelerator design, performance prediction, task off-loading, and graph classification/regression.

PYXIS is open-source, and we are constantly incorporating new accelerator designs and performance statistics to PYXIS. Our future works include:

- Incorporating more accelerator designs and performance data to PYXIS. We are developing and evaluating accelerators for many other sparse applications, such as sparse-matrix vector multiplication and graph neural networks.
- Adding more labels. Besides latency and throughput, we plan to add more labels, such as power and energy consumption. The labor (hours) for designing an accelerator is another attractive label we are considering.
- Issuing calls for accelerators and results. We will call for contributions from the whole community to contribute to PYXIS. Our standard is open-source and reproducible.
- Enriching the accelerator descriptions. We currently accept HLS C/C++ codes as the description of accelerators. However, we will consider accepting accelerator descriptions in other formats.

## ACKNOWLEDGMENT

This work is supported in part by the NSF RTML Program (CCF-1937599), CDSC industrial partners[1], and the Xilinx XACC Program.

[1] https://cdsc.ucla.edu/partners

## 6. REFERENCES


[1] Gordon E. Moore, "Cramming More Components onto Integrated Circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

[2] Jason Cong et al., "Accelerator-Rich Architectures: Opportunities and Progresses," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2014, pp. 1–6.

[3] Jason Cong et al., "Customizable Computing—From Single Chip to Datacenters," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 185–203, 2018.

[4] John L. Hennessy and David A. Patterson, "A New Golden Age for Computer Architecture," *Communications of the ACM*, vol. 62, no. 2, pp. 48–60, 2019.

[5] William J. Dally et al., "Domain-Specific Hardware Accelerators," *Communications of the ACM*, vol. 63, no. 7, pp. 48–57, 2020.

[6] Jie Wang, Licheng Guo, and Jason Cong, "AutoSA: A Polyhedral Compiler for High-Performance Systolic Arrays on FPGA," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 93–104.

[7] Jason Cong and Jie Wang, "PolySA: Polyhedral-Based Systolic Array Auto-Compilation," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[8] Chen Zhang et al., "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2015, pp. 161–170.

[9] Yunji Chen et al., "DaDianNao: A Machine-Learning Supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 609–622.

[10] Yu-Hsin Chen et al., "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[11] Norman P. Jouppi et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 1–12.

[12] Samuel Williams et al., "Roofline: An Insightful Visual Performance Model for Multicore Architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[13] Tae Jun Ham et al., "Graphicionado: A High-Performance and Energy-Efficient Accelerator for Graph Analytics," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.

[14] Junwhan Ahn et al., "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 2015, pp. 105–117.

[15] Guohao Dai et al., "FPGP: Graph Processing Framework on FPGA A Case Study of Breadth-First Search," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 105–110.

[16] Shijie Zhou et al., "HitGraph: High-throughput Graph Processing Framework on FPGA," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2249–2264, 2019.

[17] Linghao Song et al., "GraphR: Accelerating Graph Processing Using ReRAM," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 531–543.

[18] Mingxing Zhang et al., "GraphP: Reducing Communication for PIM-Based Graph Processing with Efficient Data Partition," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 544–557.

[19] Timothy A. Davis and Yifan Hu, "The University of Florida Sparse Matrix Collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, pp. 1–25, 2011.

[20] John L. Henning, "SPEC CPU2006 Benchmark Descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[21] Shuai Che et al., "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*. Ieee, 2009, pp. 44–54.

[22] Vijay Janapa Reddi et al., "MLPerf Inference Benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 446–459.

[23] Jason Cong et al., "Understanding Performance Differences of FPGAs and GPUs," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2018, pp. 93–96.

[24] Yuan Zhou et al., "Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software Programmable FPGAs," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2018, pp. 269–278.

[25] Yuwei Hu et al., "GraphLily: Accelerating Graph Linear Algebra on HBM-Equipped FPGAs," in *2021 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[26] Linghao Song et al., "Sextans: A Streaming Accelerator for General-Purpose Sparse-Matrix Dense-Matrix Multiplication," *arXiv preprint arXiv:2109.11081*, 2021.

[27] Atefeh Sohrabizadeh et al., "AutoDSE: Enabling Software Programmers Design Efficient FPGA Accelerators," *arXiv preprint arXiv:2009.14381*, 2020.

[28] Nan Wu et al., "IronMan: GNN-assisted Design Space Exploration in High-Level Synthesis via Reinforcement Learning," *arXiv preprint arXiv:2102.08138*, 2021.

[29] Atefeh Sohrabizadeh et al., "Enabling Automated FPGA Accelerator Optimization Using Graph Neural Networks," *arXiv preprint arXiv:2111.08848*, 2021.

[30] Ashish Vaswani et al., "Attention Is All You Need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[31] Yunsheng Bai et al., "SimGNN: A Neural Network Approach to Fast Graph Similarity Computation," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019, pp. 384–392.

[32] Zongyue Qin et al., "GHashing: Semantic Graph Hashing for Approximate Similarity Search in Graph Databases," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 2062–2072.