Titre: Title:	Studying the Practices and Challenges of Developing Hardware Description Language Programs
Auteur: Author:	Fatemeh Yousefifeshki
Date:	2022
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Yousefifeshki, F. (2022). Studying the Practices and Challenges of Developing Hardware Description Language Programs [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/10340/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Studying the Practices and Challenges of Developing Hardware Description Language Programs

FATEMEH YOUSEFIFESHKI
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Mai 2022

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Studying the Practices and Challenges of Developing Hardware Description Language Programs

présenté par **Fatemeh YOUSEFIFESHKI**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Jinghui CHENG, président
Foutse KHOMH, membre et directeur de recherche
Heng LI, membre et codirecteur de recherche
Maxime LAMOTHE, membre

DEDICATION

To my family, specially my parents, who always supported me...

ACKNOWLEDGEMENTS

This dissertation is supervised by Dr. Foutse Khomh and co-supervised by Dr. Heng Li. I would like to express my most sincere appreciation to Dr. Khomh and Dr. Li for their patience, wise comments, advice, precious assistance, and valuable support throughout my Master's program. I would like to thank my committee members, Dr. Jinghui Cheng, and Dr. Maxime Lamothe for evaluating my master's thesis. Besides all the support from my advisors, I cannot ignore my family's unconditional love and generous support. I would like to dedicate my dissertation to my amazing father and my mom for her continued love.

RÉSUMÉ

Avec la fin de la loi de Moore et de la mise à l'échelle de Dennard, les architectures à usage général (par exemple, les CPU RISC) deviennent insuffisantes pour répondre à nos besoins croissants en matière de calcul haute performance. Les architectures spécifiques à un domaine (par exemple, le TPU de Google) sont optimisées pour des applications spécifiques et peuvent donc atteindre de meilleures performances. Le développement de telles architectures spécifiques à un domaine nécessite généralement l'écriture de programmes dans des langages de description du matériel (HDL). Par rapport aux langages de programmation polyvalents (GPPL) traditionnels (par exemple, C++, Java, Python), le développement de programmes en HDL (par exemple, VHDL ou Verilog) a actuellement peu de soutien de la communauté de génie logiciel. Un tel déséquilibre dans le soutien aux GPPL et aux HDL entravera les progrès futurs des systèmes informatiques. L'une des raisons de ce déséquilibre semble être la méconnaissance des défis posés par les langages de description du matériel. Afin d'améliorer cette situation, dans ce mémoire, nous faisons une première tentative d'élucidation de ces défis, en étudiant les pratiques et les défis du développement de programmes en HDL, en examinant deux sources de données liées au développement de programmes HDL : 1) les forums techniques où les développeurs posent des questions relatives aux HDL, et 2) les projets open-source où les développeurs écrivent du code HDL.

Les développeurs posent généralement des questions techniques et y répondent sur des plateformes tels que les forums Stack Exchange. Ces questions peuvent communiquer les défis rencontrés par ces derniers lors du développement de programmes HDL. Par conséquent, notre première étude identifie et analyse plus de 16 000 questions relatives au HDL provenant de deux forums Stack Exchange : Stack Overflow (SO) et Electrical Engineering (EE). Nous analysons ces questions en combinant une analyse qualitative et une modélisation automatique des sujets, afin de comprendre les défis auxquels sont confrontés les développeurs lors du développement de programmes HDL.

Ces dernières années, l'accélération matérielle des réseaux neuronaux profonds (DNN) est devenue un domaine d'application populaire de la co-conception matériel-logiciel spécifique à un domaine. Par conséquent, dans notre deuxième étude, nous examinons 321 projets open-source liés à l'accélération matérielle pour les réseaux neuronaux profonds, afin de mieux comprendre les pratiques et les défis du développement HDL dans les projets du monde réel. Nous étudions les catégories de projets qui exploitent l'accélération matérielle pour les DNN, la distribution des langages de programmation et les profils des développeurs de ces

projets, ainsi que les objectifs de l'utilisation de l'accélération matérielle pour les DNN dans ces projets.

Notre travail a mis en évidence les différences entre le développement de programmes HDL et GPPL, et a permis d'identifier des possibilités d'amélioration pour le développement HDL. Nous espérons que notre travail contribuera à sensibiliser la communauté du génie logiciel sur la nécessité de soutenir le développement HDL, afin d'éviter que les composants HDL ne deviennent des goulots d'étranglement dans le développement de futurs systèmes informatiques spécifiques à un domaine.

ABSTRACT

Modern computer systems typically consist of two indispensable components: general-purpose hardware (e.g., CPU) and highly-versatile software (e.g., web applications). In this context, Turing award winners John Hennessy and David Patterson, in their recent Turing lecture, declared that computer architectures are at the point of shifting from a general-purpose approach to a domain-specific hardware-software co-design approach. Developing such domain specific architectures typically requires writing programs in Hardware Description Languages (HDLs). This work makes an initial attempt to understand the practices and challenges of developing HDL programs and explore opportunities for the Software Engineering community to support the practices of HDL program development, by examining two sources of data related to HDL program development: 1) technical forums where developers ask HDL-related questions, and 2) open-source projects where developers write HDL code.

Developers usually ask and answer technical questions on technical forums such as Stack Exchange forums. These questions may communicate challenges faced by developers when developing HDL programs. Therefore, our first study identifies and analyzes over 16,000 HDL-related questions from two Stack Exchange forums: Stack Overflow (SO) and Electrical Engineering (EE) Stack Exchange. We analyze these HDL-related questions using a combination of qualitative analysis and automated topic modelling, in order to understand the challenges faced by developers when developing HDL programs.

In recent years, hardware acceleration for Deep Neural Networks (DNNs) have become a popular application area of domain-specific hardware-software co-design. Therefore, in our second study, we examine 321 open-source projects that are related to hardware acceleration for DNNs, to further understand the practices and challenges of HDL development in real-world projects. We study the categories of projects that leverage hardware acceleration for DNN, the distribution of programming languages and developer profiles of these projects, as well as the purposes of using hardware acceleration for DNN in these projects.

Our work highlights the differences between developing HDL programs and general-purpose programming language (GPPL) programs and identified opportunities for improving HDL development. We expect our work to draw the software engineering community's attention to supporting HDL development and prevent HDLs from becoming the bottleneck in developing future domain-specific computer systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

HDL	Hardware Description Language
GPPL	General-purpose programming language
Q&A	Questions and Answers
SO	Stack Overflow
EE	Electrical Engineering Stack Exchange
DNN	Deep Neural Network
ML	Machine Learning
API	Application Programming Interface
CNN	Convolution Neural Network
ANN	Artificial Neural Network
BNN	Binary Neural Network
SNN	Spiking Neural Network
RNN	Recurrent Neural Network
GPU	Graphic Processing Unit
CPU	Central Processing Unit
TF	TensorFlow
UI	User Interface
RISC	Resources Information Standards Committee
TPU	Cloud Tensor Processing Units

CHAPTER 1 INTRODUCTION

Modern computer systems typically consist of two indispensable components: general-purpose hardware (e.g., CPU) and highly-versatile software (e.g., web applications). However, with Moore's law and the Dennard scaling ending, it has been increasingly difficult to increase the number of transistors on a chip, which poses critical challenges to enhancing the computing power of general-purpose architectures (e.g., RISC) [2–4]. In this context, Turing award winners John Hennessy and David Patterson, in their recent Turing lecture, declared that computer architectures are at the point of shifting from a general-purpose approach to a domain-specific hardware-software co-design approach [2]. Domain-specific architectures (e.g., Google's TPU, Apple's Neural Engine, or Microsoft's FPGAs for deep neural networks) are more closely tailored to the application domain and can provide new opportunities to improve the performance of computer systems. Developing such domain-specific architectures typically requires writing programs in Hardware Description Languages (HDLs) [3]. However, there exists an imbalance between the support for HDL program development and that for developing programs in traditional general-purpose programming languages (GPPLs) (e.g., C++, Java, Python) [2–4]. We believe that such an imbalance in GPPLs and HDLs will impede future advances in computer systems, particularly in domain-specific applications. This work makes an initial attempt to understand the practices and challenges of developing HDL programs and explore opportunities for the Software Engineering community to support the practices of HDL program development. This work is organized through two perspectives: 1) mining technical forums where developers ask HDL-related questions to understand the challenges of developing HDL programs, and 2) mining open-source projects where developers write HDL code, in particular, projects that leverage hardware acceleration for DNN, to understand the practices of writing HDL in real-world projects.

1.1 Studying the challenges of developing HDL programs by mining technical forums

Hardware description languages (**HDLs**) are programming languages with particular capabilities for describing the concurrent nature of digital logic and electronics. On the time and space dimensions, HDLs represent the structure and behavior of hardware. VHDL [5] and Verilog [6] [7] are the two most widely used HDLs in reality applications. They are now the bridges every design needs to traverse to access synthesis and implementation technologies, independent of whatever HDL was used to generate it. Any HDL-capable programming lan-

guage may be isolated from lower-level stages and tools by producing Verilog or VHDL code from its synthesizable subset. Hardware designers are always searching for innovative ways to save design time and expenses [3]. The fact that the hardware ecosystem is a monoculture of a few chip designs from a few manufacturers limits design productivity. An ecosystem of readily available intellectual property blocks that may be combined to generate new chip designs is critical for the hardware industry. Improvements in HDLs will help this transition. Modern high-level programming languages include several features that support modularity and abstraction. A type system, for example, may restrict data at an interface [8]. However, most hardware description languages (HDLs) only offer a few of these features. We use the term wires to describe the module's interface, which may be further refined to input or output. Hundreds of designs revealed that these interfaces have complex needs, not only in terms of data use and processing but also in terms of compositions that contribute to well-defined digital designs [8]. These languages lack strong abstraction capabilities seen in contemporary software languages, reducing designer productivity by making component reuse difficult. This new development paradigm raises concerns about how hardware developers should build and treat HDL implementations. This work makes an initial attempt to understand the challenges of developing HDL programs and explore opportunities for the SE community to support the practices of HDL program development. Developers usually ask and answer technical questions on technical forums such as Stack Exchange forums[1]. These questions may communicate challenges faced by developers when developing HDL programs. Therefore, this work identifies and analyzes over 16, 000 HDL-related questions from two Stack Exchange forums: Stack Overflow (SO) and Electrical Engineering (EE) Stack Exchange. We analyze these HDL-related questions using qualitative analysis and automated topic modeling to understand developers' challenges when developing HDL programs. Our study is organized by answering the following three research questions (RQs):

RQ 1.1: What types of questions do developers ask about HDL?

We manually examined a statistically significant representative sample of HDL-related questions posted in the SO and EE forums to understand the types of information sought by developers about HDL program development. We discovered ten types of questions by expanding on a prior classification [1]. Our findings indicate the need for more resources and tools to aid developers in understanding, verifying and diagnosing issues in HDL programs.

RQ 1.2: What are the topics in the HDL-related questions?

We use topic models to derive the semantic topics included in developers' questions to under-

[1]https://stackexchange.com

stand developers' challenges in developing HDL programs. We observe that HDL program development shares most of the challenges (e.g., *syntax errors* and *file I/O*) of GPPL program development. However, some challenges (e.g., lower level operations such as *bit operations* and *register operations*, or synchronization through *clocked processes*) are only notable for HDLs.

RQ 1.3: What questions are difficult for developers to get answers?

We estimate the difficulty level of HDL-related questions by determining whether and how fast a question receives an accepted answer. We observe that HDL-related questions are less likely and take longer than questions related to GPPLs to get accepted answers. Besides, the most difficult topics (e.g., *file/memory IO*, *data transfer*, *state machines*, *syntax errors*, and *testing and simulation*) are consistent across different languages.

1.2 Studying the practices of developing HDL programs by mining open-source projects that leverage hardware acceleration for DNN

Deep Neural Networks (DNNs), also known as Deep Learning, are a subset of AI, which, according to John McCarthy, the computer scientist who invented the phrase in the 1950s, is "the science and engineering of constructing intelligent computers that can achieve goals as people do". DNNs are the cutting-edge solutions for various applications, including computer vision, speech recognition, and natural language processing, among others. Artificial Neural Networks (ANNs) are a mathematical architecture that connects a vast number of essential elements known as neurons, each of which can make simple mathematical decisions [9]. An external neural network has only three layers: the input layer, one hidden layer, and the output layer. As the number of hidden layers in a neural network grows, it becomes a Deep Neural Network (DNN). As a result, Deep Learning can be thought of as a subset of Artificial Neural Networks with many processing layers. They are more precise and continue to improve as more neuron layers are added. Feed-Forward Neural Network, Recurrent Neural Network (RNN), and Convolutional Neural Network (CNN) are three popular Deep Neural Network models [9].

However, DNNs' increased accuracy comes at the cost of significant computational complexity. While general-purpose compute engines, particularly graphics processing units (GPUs), have been the basis for much DNN processing, there is growing interest in more specialized DNN acceleration. Over the last two decades, many DNN models have been built. A DNN model is typically very complex and takes long time and expensive resources to perform training and inferences. Thus, researchers and practitioners have proposed various approaches to

make DNN model training and inferences more efficient. These approaches typically fall into two categories: 1) model optimization (the software way) and 2) hardware acceleration (the hardware way). There has been recent evidence from the field of deep learning that FPGAs (Field-Programmable Gate Arrays) are an important factor in the acceleration of DNNs (Deep Neural Networks).An HDL (Hardware Description Language) vendor tool is used to synthesize DNN from a high-level language such as python after manual transformation. HDL expertise is required for this transition, which limits the use of FPGAs [10]. In data centers, FPGAs are now widely used to offload GPU- and CPU-based inference engines. Starting with targeted FPGAs, model development and optimization frameworks, and an ecosystem of supported libraries, we are still in the early stages of defining, expanding, and deploying such capabilities. Over the next five years, FPGA capabilities are expected to rapidly improve, allowing them to tackle a wide range of real-world applications [9]. In this thesis, we investigate hardware acceleration for DNN projects to answering the following research questions (RQs):

RQ 2.1: What are the categories of the projects that leverage hardware acceleration for DNN?
To determine the type of models used in the projects, we manually examined 321 projects linked to hardware acceleration for DNN on GitHub. We also look at the kind of tasks that may be aided by employing hardware acceleration.

RQ 2.2: What are the distributions of programming languages and developer profiles in the projects leveraging hardware acceleration for DNN?
We extracted the programming languages used in the DNN hardware acceleration projects and determine the portions of code corresponding to HDL and GPPL languages. In addition, we collected and analyzed information about developers' activities to identify their area of expertise.

RQ 2.3: What are the purposes of using hardware acceleration in the projects leveraging hardware acceleration for DNN?
The description of 321 DNN hardware acceleration projects were manually inspected in order to identify the purpose of using hardware acceleration in these projects.

1.3 Contributions of the Thesis

In summary, this thesis makes the following contributions:

- We observe that developing HDL programs face similar challenges as GPPL and identified challenges specific to HDLs (e.g., lower level operations such as bit and register operations or synchronization through clocked processes).
- We also observe that it is more challenging for developers of HDL programs to receive community support than GPPL (i.e., HDL-related questions are less likely and take longer to get accepted answers), which calls for efforts to enhance developer engagement or platform support in answering questions in technical forums (e.g., SO).
- Our work identified opportunities for the research community to improve the practices of developing HDL programs, such as to improve the language abstraction for handling bit/register operations or file/memory I/Os, provide actionable information to address syntax errors, and–or to develop tools to improve testing and simulation.
- We investigate an application domain of hardware-software co-design: DNN hardware acceleration, and the practices of real-world open source projects in this domain. Our results provide insights for practitioners and researchers to understand the practices of DNN acceleration as well as the practices and challenges of hardware-software co-design in this context.

1.4 Organization of the Thesis

The remainder of this thesis is organized as follows: Chapter 2 introduces the concepts and background that help to understand our research work related to hardware description languages and DNN hardware acceleration. Chapter 3 examines the challenges of developing Hardware Description Language Programs. Chapter 4 examines DNN hardware acceleration practices in GitHub projects. Finally, Chapter 5 summarizes and concludes the thesis, and discusses future work.

CHAPTER 2 BACKGROUND AND LITERATURE REVIEW

2.1 Background

We first introduce some background related to our work, including the background on hardware description languages and hardware acceleration for DNN.

2.1.1 Background on Hardware Description Languages (HDL)

Hardware Description Languages (HDLs) are programming languages used to describe the structures and behaviors of digital logic circuits, which are the essential components of modern computer systems [11]. As illustrated in Figure 2.1, there are two types of digital logic circuits: combinational logic circuits and sequential logic circuits. The inputs fully determine the outputs of a combinational logic circuit. Examples of the combinational logic includes the basic *and*, *or*, and *xor* gates and calculators for which outputs only depends on the inputs. The outputs of a sequential logic circuit are determined by both the inputs and a previous state. Examples of sequential logic include flip-flops and register circuits. A sequential logic circuit needs a mechanism to control the timing of updating the state, either synchronously or asynchronously; thus, sequential logic is further divided into synchronous and asynchronous logic. The synchronous logic synchronizes the state changes by using a clock signal, while the state changes in the asynchronous logic can occur in response to changes in the inputs.

HDLs typically support both combinational logic and sequential logic [3]. Thus, the most significant difference between HDLs and GPPLs is the explicit inclusion of the notion of time in HDLs [11]. The most popular HDLs include Verilog [12], VHDL [13], and SystemVer-

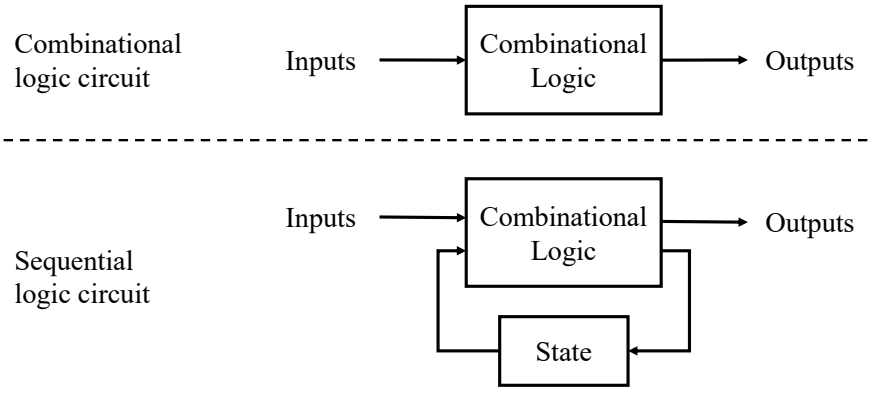

Figure 2.1 Illustration of combinational logic circuits and sequential logic circuits

ilog [14]. Verilog is the dominant HDL adopted by practitioners [3]. Verilog can describe, verify, and simulate digital logic circuits its structure is inspired by C. It possesses most of the same features as C, including a preprocessor, control flow, and operators. SystemVerilog is an enhanced version of Verilog that extends Verilog by adding a rich and robust type system, especially for user-defined systems, while still keeping weak typing for the built-in Verilog types [15]. System-Verilog also extends Verilog by providing capabilities for interface abstraction and packaging [15]. VHDL is another popular HDL, providing similar functionalities as Verilog. However, VHDL is derived from the Ada programming language, and it is firmly and richly typed. VHDL is usually more verbose than Verilog [15].

2.1.2 Background on Hardware Acceleration for DNN

Deep Neural Networks (DNN) have recently proved to be one of the most powerful machine learning methods. DNNs owe their efficacy to their ability to learn complex models. Thus, researchers and practitioners have proposed various approaches to make DNN model training and inferences more efficient. There are two main approaches to improving DNN methods. The model optimization methods are used in the first approach, which focuses on software solutions. The Tensorflow Model Optimization Toolkit [16] is one of the tools in software-based approaches with a concentration on aspects like post-training quantization, weight clustering, quantization aware training, etc. [16]. The second approach, which we will focus on in this study, is hardware-based methods concentrating on hardware acceleration. Hardwares (including FPGA) are typically designed/programmed by hardware description languages (HDLs) such as Verilog. A multi-language system that provides a framework for developing solutions to these intellectual problems is presented by the authors of [2,3] in their discussion of the challenges that researchers interested in hardware description language, compilers, and formal methods face. They identify opportunities to apply programming language techniques to address productivity issues in hardware design. One of the best-known solutions in this category is Microsoft Azure FPGAs [17]. With Microsoft Azure FPGAs, developers will be able to deploy machine learning models to field-programmable gate arrays (FPGAs).

2.2 Literature Review

In this section we review the related literature on domain-specific architectures, HDL development, technical forum analysis, and hardware acceleration for DNN.

2.2.1 Domain-Specific Architectures

With Moore's law and the Dennard scaling ending, general-purpose architectures (e.g., RISC CPU) are becoming insufficient to handle our growing needs for high-performance computing [2–4]. According to Turing award winners John Hennessy and David Patterson, computer architectures are at the point of shifting from a general-purpose approach to a domain-specific hardware/software co-design approach [2]. Domain specific architectures (e.g., Google's TPU, Apple's Neural Engine, or Microsoft's FPGAs for deep neural networks), trading off specificity for flexibility, are more closely tailored to the application and thus can achieve (order(s) of magnitude) higher efficiency [18,19]. For example, prior work has proposed domain-specific architectures to accelerate the training and inference of deep neural network models [20–24]. In addition, domain-specific architectures have found applications in a wide range of other emerging domains that require intensive and efficient computations, such as security [25], image processing [26, 27], simulation [28], bioinformatics [29], and big data [30]. Developing such domain specific architectures typically requires writing programs in HDLs which are then compiled into specifications of hardware units (e.g., transistors) and their interconnections [3]. **The increasing demand and applications of domain-specific architectures motivate us to study the challenges and opportunities in HDL program development.**

2.2.2 HDL Development

Prior work improves the practices of HDL development from two aspects: 1) HDL design and 2) HDL compilers and transpilers. The first line of work aims to improve the design of HDL. In particular, recent studies have explored the opportunities of leveraging the advances in GPPLs (e.g., meta-programming) to improve HDL [7, 31–38]. For example, *Chisel* [31] and *Magma* [32] are HDLs based on the GPPLs Scala and Python, respectively, by embedding hardware units in these GPPLs through meta-programming. The HDL *SystemC* [39] is derived from C++ by providing a set of classes and macros for system-level design, modeling, and verification. *ASystemC* [37] extends *SystemC* by adding the features of Aspect Oriented Programming (AOP). Truong and Hanrahan [3] propose a vision for future HDLs that is based on a meta-programming language, an embedded HDL, and a variety of domain-specific languages.

Another line of work targets HDL compilers [40–45] and transpilers [36, 46–48]. To improve the development of compilers, prior studies have developed common infrastructures (e.g., intermediate representations, or IR) for HDL compilers [40, 41, 43–45]. Prior studies have also developed approaches to transform GPPLs to HDLs or vice-versa (i.e., transpilers).

For example, Quokka [47] is a transpiler that can transform C# to HDLs. In contrast, Verilator [46] can transforms Verilog and SystemVerilog to C++. **Despite these efforts in HDL language development, prior work did not investigate the practical issues faced by HDL developers. Therefore, this work examines the questions and answers in technical forums such as SO to understand developers' challenges when developing HDL programs.**

2.2.3 Technical Forum Analysis

Extensive studies have been performed to study posts in technical forums, in particular, the Stack Exchange forums [49]. Here we focus on the studies that are most relevant to our work: 1) studies that analyze the categories of questions and 2) studies that analyze the topics of questions.

Categorization of technical forums questions.

Prior studies have investigated the categories of technical forum questions [1, 50–54].

For example, Rosen and Shihab [51] manually studied a random sample of 384 SO questions related to mobile development and derived four categories of questions: *why*, *how*, *what*, and *other*. Beyer et al. [1] surveyed these studies [50–54] and summarized a categorization of SO questions, including seven categories of questions: *API usage*, *discrepancy*, *errors*, *review*, *conceptual*, and *API change*. In this work, we study the categories of HDL-related questions on two Stack Exchange forums, and we base on the categorization of the work of Beyer et al. [1].

Topic analysis of technical forum posts.

Topic models are used extensively in prior work to understand the topics of general technical forum posts and the topic trends [55], [56], [50]. Prior work also leverages topic models to understand the topics of technical forum posts related to specific application development domains, such as mobile application development [51,57], client application development [58], concurrent programming [59], security-related development [60], big data development [61], Swift programming [62], IoT development [63], and deep learning development [64]. In addition, prior work leverages topic models to understand non-functional requirements communicated in technical forum posts [65, 66]. Zhang et al. [67] use topic models to detect duplicate questions in technical forums. Finally, Treude et al. [68] proposes an automated approach to suggest configurations of topic models for analyzing technical forum data. Most of these studies use the Latent Dirichlet Allocation (LDA) algorithm or its variants to extract topics from the text of the technical forum posts. In this work, we also leverage the LDA

algorithm to extract topics from technical forum posts related to HDLs. **The increasing demand for domain-specific architectures calls for our emphasis on HDL program development. However, no work exists that studies the practical issues faced by HDL developers. Therefore, this work makes an initial effort to study the challenges of HDL program development and identify opportunities to address these challenges.**

2.2.4 Hardware acceleration for DNN

Hardware acceleration refers to the technique of moving a complex computing operation from the CPU to specialized hardware in order to achieve greater efficiency than would be possible with a traditional software implementation of the same task. Better models and a larger dataset are needed to get real-time results that are as precise as possible. Taking time to make a decision is also vital. Deep Learning models become increasingly complicated as they evolve. A large number of procedures and parameters, as well as increased computational resources, are required as a result of this expansion. GPUs, ASICs, and FPGAs are examples of hardware accelerators [9]. The Versatile Tensor Accelerator (VTA) is a deep learning accelerator that is open, generic, and adaptable, according to the authors of [69]. Brandon Reagen et al. [70] have introduced Minerva, a highly automated co-design technique to improve DNN hardware accelerators across the algorithm, architecture, and circuit levels. Authors of [71] have provided a comprehensive comparison of model compression methods and hardware acceleration methods. They describe various hardware platforms used for DNNs and explain different optimizations applied to improve throughput.

To speed up the simulation of ResNet networks using approximate multipliers, Filip Vaverka et al. [72] developed a Tensor Flow extension TFApprox. Vojtech et al. [73] demonstrate how a large number of approximate multipliers can be applied to do a resilience study of a ResNet DNN hardware accelerator and to choose the best approximate multiplier for a specific application. According to Charles Mackin and colleagues [74], a methodology for selecting the best hardware circumstances under which to program weights, and its usefulness in the face of substantial NVM variability, is examined via simulations. For wearable devices, Johnson Loh et al. [75] designed a DSP (digital signal processing) accelerator for ECG data categorization. An infrastructure for analyzing and exploring the architectural design space of deep neural network (DNN) accelerators has been developed by Angshuman Parashar et al [76]. A research by Siva Kumar Sastry Hari et al. [77] examines the propagation of soft mistakes in DNN systems and proposes two effective protective measures to reduce their influence and increase dependability. Using a cycle-accurate simulator called SCALE-SIM,

Ananda Samajdar et al. [78] provide an analytical methodology for determining the best scale-up vs. scale-out ratio for DNN inference given hardware restrictions. Maria I. Mera Collantes et al. [79] proposed a Safe-TPU implementation with high parallelism and reuse of current resources already deployed in the baseline DNN accelerator, as well as additional protocol innovations that considerably decrease the space and time necessary to construct proofs.

FPGA-based accelerators are more popular than GPU and ASIC-based accelerators due to their high performance, customizable and flexible design, high throughput with large parallelism, faster data transfer, and high speed of a DNN implemented on them [9]. Storage, computational resources, and memory bandwidth are all concerns that must be addressed when DNNs are implemented on FPGAs. The accuracy, latency, throughput, and portability of the implementation are all used to evaluate the mapping of DNNs into FPGAs [10]. Some of the effort in this field is centered on developing automated tools for rapidly deploying DNNs to FPGAs. X. Wei et al. [80] the design of a framework using systolic arrays to accelerate DNN inference. To ensure diverse network setups, the framework adopts a uniform RTL design for CONV layers and runs software in the host CPU [81]. However, the Fully-connected (FC) layers are not implemented on FPGA. Other frameworks for autonomously mapping DNNs onto FPGAs using RTL [82, 83] or RTL-HLS [84] templates have been proposed. Zhang et al. [85] have developed DNNBuilder, a tool for automatically developing high-performance DNN hardware accelerators on FPGAs to bridge the gap between quick DNN generation in software (e.g., Caffe, TensorFlow) and sluggish hardware implementation. FPGA-based accelerators are more cost-effective than GPU and ASIC-based accelerators. Noronha et al. provide an open-source tool-flow for transferring DNN numerical computing models developed in Tensorflow to FPGA [86]. V. Sze and A. Shawahna et al. [87, 88] provide tutorials and surveys of hardware acceleration for DNN. There are also many open-source projects on FPGA-based model acceleration [89].

Prior work proposed different approaches that use hardware acceleration to improve DNN model performance. However, no prior work has studied how hardware acceleration is used in real-world DNN projects. Thus, this work studies open-source GitHub projects that uses hardware acceleration for DNN, to understand the practices and challenges of DNN hardware acceleration.

CHAPTER 3 STUDYING THE CHALLENGES OF DEVELOPING HARDWARE DESCRIPTION LANGUAGE

The content of this chapter is currently under review for publication in the Information and Software Technology Journal[1]*.*

Context: As Moore's law and the Dennard scaling ended, general-purpose architectures (e.g., RISC CPU) are becoming insufficient to handle our growing needs for high-performance computing. Domain specific architectures (e.g., Google's TPU) are more closely tailored for specific applications and thus can achieve better performance. Developing such domain specific architectures typically requires writing programs in Hardware Description Languages (HDLs). Compared to traditional general-purpose programming languages (GPPLs) (e.g., C++, Java, Python), developing programs in HDLs (e.g., VHDL or Verilog) lacks support from our community. Such an imbalance in the support for GPPLs and HDLs will impede future advances in computer systems.

Objective: We believe that our software engineering community should pay more attention to supporting HDL development. Thus, we make an initial attempt in this direction to study the challenges of developing programs in HDLs by mining HDL-related questions in technical forums.

Method: We identified 16,700 HDL-related questions in two Stack Exchange forums: Stack Overflow (SO) and Electrical Engineering (EE) Stack Exchange. We examined the types of questions, the questions' topics, and identify the most challenging topics for developers.

Results: We identified ten types of HDL-related questions, including seven types identified in prior work and three new types more relevant to HDLs (e.g., questions related to code explanation and tool search). We also observed that most of the challenges facing HDL developers are similar to those facing GPPL developers, while some challenges (e.g., lower-level operations such as bit and register operations) are more specific to HDLs. Finally, we observed that HDL-related questions are less likely and take a longer time to get accepted answers than GPPL-related questions, and identified the most challenging topics of questions (e.g., file/memory I/O).

Conclusion: Our work identified opportunities for the research community to improve the practices of developing HDL programs, such as to improve the language abstraction of handling bit/register operations or file/memory I/Os, to provide actionable information to ad-

[1]Fatemeh Yousefifeshki, Heng Li, Foutse Khomh, Studying the Challenges of Developing Hardware Description Language Programs, Information and Software Technology, 2022 (under review).

dress syntax errors, or to develop tools to improve testing and simulation. We hope that our work can draw the software engineering community's attention to HDLs and prevent HDLs from becoming the bottleneck in developing future computer systems.

3.1 Introduction

Modern computer systems typically consist of two indispensable components: general-purpose hardware (e.g., CPU) and highly-versatile software (e.g., web applications). Software talks to hardware through a concise vocabulary called an instruction set architecture (ISA) (e.g., the reduced instruction set computer or RISC) [2]. Compilers (e.g., GCC) convert programs written in high-level programming languages (e.g., C++) into machine-readable programs based on the ISA-defined vocabulary. Advances in ISA and compilers allow modern computer systems to achieve highly-diverse functionalities through general-purpose hardware and highly-customizable software applications.

However, with Moore's law and the Dennard scaling ending, it has been increasingly difficult to increase the number of transistors on a chip, which poses critical challenges to enhancing the computing power of general-purpose architectures (e.g., RISC) [2–4]. In this context, Turing award winners John Hennessy and David Patterson, in their recent Turing lecture, declared that computer architectures are at the point of shifting from a general-purpose approach to a hardware/software co-design approach [2]. Domain specific architectures (e.g., Google's TPU, Apple's Neural Engine, or Microsoft's FPGAs for deep neural networks) are more closely tailored to the application domain and can provide new opportunities to improve the performance of computer systems. Developing such domain specific architectures typically requires writing programs in hardware description languages (HDLs) (e.g., VHDL, Chisel, or Verilog), which are then compiled into specifications of hardware units (e.g., transistors) and their interconnections [3]. For example, Google uses Chisel to develop Edge TPU. However, there exists an imbalance between the support for HDL program development and that for developing programs in traditional GPPLs (e.g., C++, Java, Python) [2–4]. We believe that such an imbalance in GPPLs and HDLs will impede future advances of computer systems, in particular, in domain-specific applications; thus, our SE community should pay more attention to supporting HDL program development.

This work makes an initial attempt to understand the challenges of developing HDL programs and explore opportunities for the SE community to support the practices of HDL program development. Developers usually ask and answer technical questions on technical forums

such as Stack Exchange forums[2]. These questions may communicate challenges faced by developers when developing HDL programs. Therefore, this work identifies and analyzes over 16, 000 HDL-related questions from two Stack Exchange forums: Stack Overflow (SO) and Electrical Engineering (EE) Stack Exchange. We analyze these HDL-related questions using qualitative analysis and automated topic modeling to understand the challenges developers face when developing HDL programs. Our study is organized by answering the following three research questions (RQs):

RQ1: **What types of questions do developers ask about HDL?**

We manually examined 376 statistically significant representative samples of HDL-related questions posted in the SO and EE forums to understand the types of information sought by developers about HDL program development. Our findings indicate the need for more resources and tools to aid developers in understanding, verifying and diagnosing issues in HDL programs.

RQ2: What are the topics in the HDL-related questions?

We use topic models to derive the semantic themes included in developers' questions to understand developers' challenges in developing HDL programs. We observe that HDL program development shares most of the challenges (e.g., syntax issues) of GPPL program development. However, some challenges (e.g., lower level operations such as bit and register operations or synchronization through clocked processes) are only notable for HDLs.

RQ3:**What questions are difficult for developers to get answers?**

We estimate the difficulty level of HDL-related questions by determining whether and how fast a question receives an accepted answer. We observe that HDL-related questions are less likely and take longer than questions related to GPPLs to get accepted answers. Besides, the most difficult topics (e.g., *file/memory IO*, *data transfer*, *testing and simulation*, and *state machines*) are consistent across different languages.

Our work highlights the challenges of HDL program development (e.g., lower level operations such as bit and register operations or synchronization through clocked processes). It identifies opportunities for the research community to improve the practices of HDL program development, such as to improve the language abstraction of handling file/memory I/Os, provide actionable information to address syntax errors, or develop tools to improve testing and simulation. We hope to draw the SE community's attention to the importance of HDL program development which may become a bottleneck in developing domain-specific systems that are critical in increasing computing power in the future. We encourage researchers and practi-

[2]https://stackexchange.com

tioners in the SE community to leverage the advanced methodology, techniques, and tools in traditional software development to improve the practices of HDL program development. To encourage and help future research, we share our replication package[3]

Chapter organization. The rest of the chapter is organized as follows. In Section 3.2 we describe the experiment setup of our study. In Section 3.3, we present and discuss our results for answering our research questions. In Section 3.4, we discuss the implications of our findings. Section 3.5 discusses threats to the validity of our findings. Finally, Section 3.6 we have a summary of the study.

3.2 Experiment Setup

This study aims to understand the challenges faced by developers during HDL program development. The perspective is that of researchers and practitioners interested in developing methodologies, techniques, and tools to support the development of HDL programs.

3.2.1 Context of the Study

To understand the challenges of developing HDL programs, we study Stack Exchange forums where developers ask and answer technical questions, including HDL-related questions. We manually examined all the Stack Exchange forums[4] to identify the ones that contain HDL-related questions. Specifically, we first read the description of each forum to identify the ones that may potentially contain HDL-related questions. Then, we searched HDL-related keywords (e.g., "VHDL", "Verilog") in these forums to confirm whether they indeed contained HDL-related questions. In the end, we found two Stack Exchange forums that contain HDL-related questions:

- Stack Overflow (SO)[5] is a technical Q&A forum for professional, enthusiast, and student programmers .
- Electrical Engineering (EE) Stack Exchange[6] is a technical Q&A forum for electronics and electrical engineering professionals, students, and enthusiasts.

This work extracts and studies the HDL-related questions in these two Stack Exchange forums to understand the challenges of developing HDL programs.

[3]Replication package: https://github.com/mahkamehy/Developing-HDL-program.
[4]https://stackexchange.com/sites
[5]https://stackoverflow.com
[6]https://electronics.stackexchange.com

Figure 3.1 shows an overview of our data extraction and analysis process. We first download the datasets of SO questions and answers using SOTorrent [90] and EE questions and answers using Stack Exchange Data Explorer [91] (Step 1).

Then, we identify the tags related to HDL and the related questions and their answers (Steps 2 and 3). A random sample of the questions is used to study the categories of HDL-related questions (RQ1). Then, we preprocess all the HDL-related questions and answers (Step 4) and use topic modeling to extract topics from these questions and answers (Step5). We analyze the topics to answer RQ2. Finally, we analyze the level of difficulty of the HDL questions related to different topics (Step 6) and answer RQ3. Below, we describe each of these steps in detail.

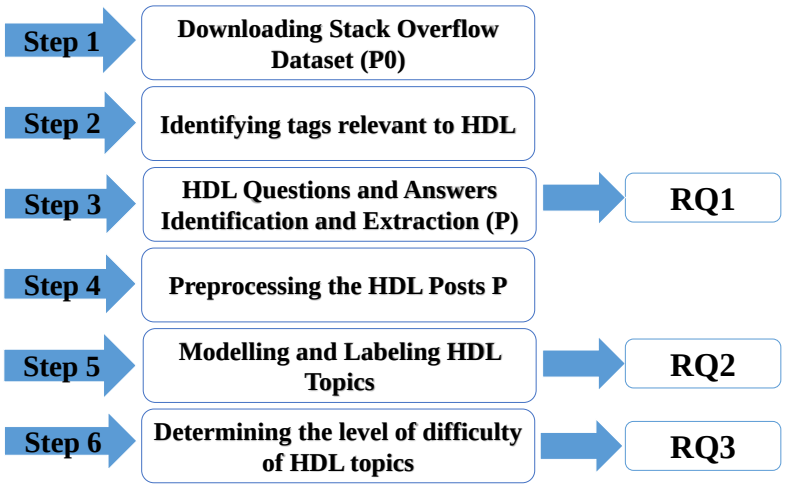

Figure 3.1 An overview of our data extraction and analysis process.

3.2.2 Data extraction and analysis steps

The SO and EE forums are selected because they include HDL-related postings and are commonly used for studying different GPPL and HDL topics [92].

Step 1 - Downloading SO Datasets.

We used SOTorrent [90] to download a dataset of SO (P_0) containing questions, their associated answers, and the associated metadata. The dataset was downloaded in May 2021.

Downloading EE datasets. We used Stack Exchange Data Explorer [91] to collect data (P_0) containing questions, answers, and metadata from the EE forum. The dataset includes questions and answers that were posted until December 2021. In both forums (SO and EE), for each question in the dataset, we obtain the question id, the title, the tag names, the

question body, the creation time of the question, the view counts, the scores, the accepted answer body (if any), and the creation time of the accepted answer (if any). Each question can have up to five tags. An answer to a question is accepted, provided that the question's creator voted it as accepted.

Step 2 - Identifying SO and EE tags relevant to HDLs.
To identify and extract HDL-related questions on both SO and EE, we need to identify a set of SO and EE tags related to HDLs. We follow the process used in prior work [93] to identify the set of HDL-related tags for both SO and EE forums. While we follow the same process to identify the HDL-related tags in the SO and EE forums, the tags of these two forums are identified separately. The initial HDL tag set T_i was derived from the most popular HDLs described in the background, i.e., tags "vhdl" , "verilog", and "syetem-verilog". Then, the three tags in T_i were used to extract from each forum's dataset P_0 the set of questions Q associated with these tags. In the next step, we extract all the tags in the question set Q to form a more extensive set of tags T_j. It should be noted that not all the tags in T_j are related to HDLs. Thus, we follow prior work [93] and use some heuristics to filter the tags in T_j based on their relationships with the initial tag set T_i. For each tag t in T_j, we use the *significance* and *relevance* scores defined below to determine if it is a HDL-related tag:

$$\text{(Significance) } S(t) = \frac{\text{\# of questions with tag } t \text{ in } Q}{\text{\# of questions with tag } t \text{ in } P_0} \tag{3.1}$$

$$\text{(Relevance) } R(t) = \frac{\text{\# of questions with tag } t \text{ in } Q}{\text{\# of questions in } Q} \tag{3.2}$$

For a tag t to be selected, the value for *significance* ($S(t)$) and *relevance* ($R(t)$) should be higher than a threshold. Based on prior work [93] and our experiments using different thresholds, we finally set the optimal threshold for S and R as 0.2 and 0.005 respectively. We further performed a manual inspection of the description and the top voted questions of each of the selected tags and identified those that are genuinely associated with HDLs: **we only keep a tag in our final set if it is associated with one HDL programming language (e.g., "vhdl")**.

Finally, our tag set T_{final} **for SO includes the tags: "vhdl," "verilog," "system-Verilog," "systemc," "chisel," "lava," "hdl," "vpi," "system-verilog-dpi," "system-verilog-assertions"**. Our final tag set T_{final} **for the EE forum includes the tags: "vhdl," "verilog," "system-verilog," and "hdl"**.

Step 3 - HDL Questions and Answers Identification and Extraction.
We used the tag set T_{final} identified in Step 2 to extract the HDL-related questions from

the SO and EE datasets P_0. In the end, we extracted 13,300 and 3,400 questions and their corresponding answers from the SO and EE forums, respectively, as our final data set P. A statistically representative 376 sample of these questions is manually examined to identify type of questions to answer our RQ1.

Step 4 - Preprocessing the HDL Posts.
In this step, the final post set P was preprocessed for topic modeling (Step 5). The process began by joining the title of each question and the question and answers' body text to generate one coherent final body. An issue with the extracted post texts was that they contained HTML tags (such as tags representing a paragraph, URL, etc.) and code snippets. Therefore, HTML tags like $< p >< /p >$ and $< a >< /a >$ and code snippets created by $< code >< /code >$ were removed from the post text. A further step was taken to remove stop words such as numbers, articles ("a," "an," and "the"), different forms of "be" ("is," "are," "been," etc.), punctuation marks, as well as symbols and non-alphabetical characters, as recognized by the MALLET's stop words list. As the final step of the process, the Porter stemmer [94] was used to clip words to their stemmed representation. We used stemmer insted of lemmatization because it can use on a large scale to improve search results. By searching not only the search phrase alone but also the word stems in the index, different word forms can be overcome and the search can also be greatly accelerated. For instance, a word like "developer" was reduced to "develop," and words like "configuration," "configure," and "configured" were all cut short to "configr."

Step 5 – Modelling and Labeling HDL Topics.
In this step, we extracted the HDL topics from the preprocessed questions and answers from the previous step (Step 4). We use the Latent Dirichlet Allocation (LDA) [95] algorithm implemented by the MALLET [96] toolkit[7]. LDA is a probabilistic topic modeling technique that models a topic based on a set of words co-occurring frequently. To enhance the quality of topic modeling results, both uni-grams and bi-grams were considered as *words* in our topic modeling process [97].

Three main hyper-parameters are typically configured to control the topic modeling results, including K, α, and β, where K controls the number of topics, α controls the topic distribution in the documents (i.e., SO posts in our context), and β controls the word distribution in the topics. Our used MALLET LDA implementation can automatically search for the optimal α and β values. We experiment with different K values (from 5 to 40 incremented by one each time) to find the best K according to a coherence score which estimates the quality of a topic by measuring the semantic similarity between the top words in the topic [98]. We

[7]Specifically, we used the Gensim wrapper of MALLET: gensim.models.wrappers.LdaMallet

use the resulting topics with the highest average coherence score as our final topics. After the automated topic modeling process generated the topics, we manually examined the resulting topics to derive a meaningful label for each topic. To assign a meaningful label to a topic, one author of this study first derived a label according to (1) the topic's top 20 keywords and (2) the top 10-15 most relevant questions associated with the topic. Then, the three authors of the study reviewed the labels together in a meeting and reassigned the labels when needed. The resulting topics from this step are analyzed to answer our RQ2.

Step 6 - Determining the level of difficulty of HDL topics
In this step, to understand how challenging and demanding the topics may be to developers, the difficulty level of each topic was estimated by two widely used heuristics [51, 52, 60, 61]: the percentages of questions of the topic receiving an accepted answer and the time taken for these questions to receive an accepted answer. Intuitively, a more complicated topic is when the related questions receive fewer accepted answers after a long waiting time. The estimated difficulty level of the topics is used to answer our RQ3.

3.3 Experiment Results

In this section, we report and discuss the results of our three research questions. We first present the motivation and approach for each research question, then discuss the results for answering the research question.

RQ1: What types of questions do developers ask about HDLs?

Motivation

As part of our effort to better understand developers' challenges, we should first discover the types of questions they are asking. This is important because it allows us to identify the types of help HDL developers seek (i.e., the areas where developers need more assistance).

In particular, we focus on the intent behind the questions asked by developers instead of the topics of the questions, similar to prior work [1].

Approach

To understand the type of HDL-related questions developers ask in the studied technical forums, we first calculate the distribution of the programming languages involved in the questions, then manually analyze a statistically representative sample of the questions to understand the developers' intent behind these questions.

Calculating the distribution of HDL programming languages in the questions. Each of the 16,700 HDL-related questions that we identified (i.e., Step 3 in Section 3.2) is associated with several tags, including at least one HDL programming language tag[8] (e.g., "vhdl"). To understand the distribution of programming languages in the HDL-related questions, we count the number of questions associated with the corresponding tag for each language. Please note that one question may be associated with more than one programming language tag.

Sampling of questions for manual analysis. To further understand the categories of developers' questions, we perform a manual analysis on a statistically representative sample of the HDL-related questions. Specifically, we randomly sample 376 questions from the two forums using a stratified sampling approach. This sample represents a 95.0% level of confidence within a 5.0% confidence interval.

Hybrid card sorting process. We adopted a hybrid card sorting process [99] to derive the categories of HDL-related questions manually. This manual analysis was based on the question categories derived in an earlier study [1]. One label was assigned to each question, and if a question is related to more than one label, which is rarely the case, the most relevant one was selected. In a joint effort, the three authors performed hybrid card sorting. The sample data were randomly divided into three equal groups of questions, and the card sorting process was done in two independent rounds, which was similar to an earlier work [100]. This method of card sorting ensures that at least two authors examine all questions.

In the first round, each author independently examined and assigned labels to one-third of the questions (i.e., one of the random groups). After that, we had discussions to ensure that all authors do a consistent labeling method. We reached a set of agreed-upon labels during these discussions. We then revisited our labels based on our consistent understanding and the agreed-upon labels.

In the second round, each author labeled another one-third of the questions, which were labeled by the other two authors in the first round. To avoid bias, each half of the one-third questions (i.e., one-sixth of the total questions) was randomly drawn from another author's previously labeled questions in the first round. We made sure that each question was labeled by two authors in the two rounds combined. We based our labeling on the results from the first round, but the authors were free to assign new labels in this round. After the individual labeling, the second round of discussions was held to analyze the second-round labeling results and verify the assigned labels' consistency. To finalize the process of labeling the

[8] As described in Section 3.2, we only kept the tags that are associated with HDL programming languages in our final tag set.

questions, each author revisited its labels according to the second-round discussions. Finally, we arranged another meeting to resolve the disagreement on the labeling results and decided on the final agreed-upon label for each question.

Results

Developers' questions are dominantly related to VHDL, Verilog, and System Verilog. Table 4.4 shows the distribution of questions related to each hardware description language. In the two studied forums, VHDL is associated with most of the questions (i.e., 42.2%), followed closely by Verilog, which is associated with 40.5% of questions, and System Verilog, which is associated with 16.7% of questions. Questions associated with at least one of these top three languages account for 93.0% of all the questions; questions associated with at least one of the top five languages account for 98.0% of all the HDL-related questions. There is a minimal number of questions associated with some less-popular languages, such as VPI and Lava, which are associated with only 15 (0.1%) and 11 (0.1%).

Table 3.1 The distributions of programming languages (tags) involved in the HDL-related questions.

Language	VHDL	Verilog	SystemVerilog	HDL	Chisel	SystemC	SystemVerilog Assertions	SystemVerilog DPI	VPI	Lava	Total
SO Questions	5,544 (41.6%)	5,346 (40.1%)	2,651 (19.9%)	819 (6.1%)	565 (4.2 %)	265 (1.9%)	158 (1.1%)	49 (0.3%)	15 (0.1%)	11 (0.1%)	13,300 (100%)
EE Questions	1,701 (43.9%)	1,616 (41.7%)	310 (8.0%)	240 (6.2%)	-	-	-	-	-	-	3,867 (100%)
Total Questions	7,245 (42.2%)	6,962 (40.5%)	2,871 (16.7%)	1,059 (6.1%)	565 (3.2%)	265 (1.5%)	158 (0.9%)	49 (0.2%)	15 (0.1%)	11 (0.1%)	17,167 (100%)

Note: Each question may be associated with more than one programming language tag.

Developers ask similar types of HDL-related questions as the questions related to GPPLs (e.g., Java). Table 3.2 describes the type of questions asked about HDLs on the SO and EE forums and their frequency. All seven types of questions identified in prior work [1] were also identified in HDL-related questions, indicating that developers ask similar types of questions across HDLs and GPPLs.

Compared to questions related to GPPLs, developers of HDLs ask more questions related to *error* and *reviews*, and fewer questions related to *API usage* and *API change*. In the 376 manual examined questions, the *error* category is the most frequently occurring category, accounting for 28.9% of all occurrences. Developers in HDL- related posts have more challenges related to code errors and compiler errors. Generally, questions in this category are associated with "why," e.g.,

Table 3.2 A taxonomy of question categories which bases on and extends [1].

Category	Description	% in SO	% in EE	% in Both
Error	When creating or running HLD programs, this category of questions looks for explanations for "why" and answers to errors and exceptions.	30.6	22.3	28.9
API usage	This kind of question is typically characterised by the phrase "how to", such as "how to utilize an API" or "how to implement a function".	19.6	13.1	18.3
Discrepancy	Typically, questions in this category seek explanations or answers for unanticipated outcomes (e.g., "what is the issue", "why does it not work").	17.0	17.1	17.0
Review	How/Why is this working? is an example of a question in this area. "Is there a better way?" In general, these inquiries seek a better solution to an issue or assistance in evaluating the existing solution.	17.0	5.2	14.6
Conceptual	The background and fundamental concept of an API are the subject of questions in this area.	10.0	27.6	13.5
Learning	This category includes questions about finding learning materials such as documentation, research papers, tutorials, and websites.	1.6	3.9	2.1
Tool search*	"I'm searching for...", "Is there a tool for...", or "Can the tool do..." are examples of questions in this area. These questions look for tools to address a particular issue or to evaluate a tool's capabilities.	1.3	3.9	1.8
Code explanation*	This category contains questions relating to describing or defining code snippets by the phrase "can someone explain" .	1.0	3.9	1.5
How to test*	This category includes questions about problem related to simulation test by the phrase "how to test"	1.3	1.3	1.3
API change	This category of questions is concerned with API changes and the resulting compatibility problems and other consequences.	0.3	1.3	0.5

New HDL question categories discovered*.

> *Any ideas **why** it would just crash? I get no indication from the program console*

API usage is the second most common category, accounting for 18.3% of all instances. The majority of inquiries in this category are identified by "how to," e.g.,

> *I would like to write a task which leverages the Zynq Verification IP (and associated API) inside it. I can't figure out how I would implement this in my testbench? I am new to SV, and am guessing that I need to pass the zynq processing system object as an argument so I can access it's API inside my super-task.*

The third and fourth most common categories are *discrepancy* and *review*. They account for 17.0% and 14.6% of all occurrences, respectively. Questions of the Review category usually show a code snippet and ask for suggestions or better solutions.

Questions of the Discrepancy category usually describe an unexpected result and ask for help. The following are examples of questions of the *discrepancy* and *review* categories, respectively:

> *This is LSFR of 10 bits. I instentiated LSFR module in verilog. You can see in the given code below. The output of LSFR is Current State. I want to access each of its individual bits. But here i am getting 0 for Current State. It is not updating. Please any one can help me.*

> *I am stuck in converting my 4 bit std vector to 5 bit. I am supposed to do logic operations and arithmetic operations with 4 bit inputs. (4 bit ALU) However, for arithmetic operations, I might have a carry out bit and I don't know how to keep it. Here is my code, I tried to define a temp 5-bit vector and make temp(4) to carry out. How can I seperate temp(4) and temp(3 downto 0) in 4-to-1 multiplexer?*

Developers of HDL ask three new types of questions, including *code explanation*, *how to test*, and *tool search*.

Phrases such as "can someone explain," "how to test," and "is there any tool" are frequently associated with these types of questions. The following is an example of a question in the *code explanation* category.

> *I just want to know the difference between this two statement. Is there a difference between both.*

The *tool search* category includes questions about finding tools, frameworks, or libraries that can help developers solve an HDL-related issue, as well as questions about validating if a tool, framework, or library can assist them in solving a problem, for instance,

> *Is there any tool out there like this? If it could even get me 90 percent of the way I'd be happy.*

This category denotes the absence of well-established tools to assist in the creation of HDL programs. The category of *how to test* indicates the challenge of testing HDL programs, for example,

> *I have been working to design a UART in vhdl How do I test it though? I have tested the transmitter using TeraTerm. Is it possible to send data using Tera Term as well? If yes, could you tell me how?*

Differences between the types of questions in SO and EE. One of the most significant differences between the questions of SO and EE is the popularity of the *conceptual* questions. The *conceptual* questions are the most popular category in the EE forum (with 27.6% questions in this category), while it is the fifth popular category in the SO forum (with 10.0% questions in this category). Another big difference between the two forums is the popularity of the *review* questions. While *review* is the third popular category in the SO forum (with 17.0% questions), it is the fifth popular category in the EE forum (within only 5.2% questions).

Our observations indicate that HDL developers tend to ask background and conceptual questions in the EE forum while asking for code reviews or recommendations in the SO forum.

The nature of the two forums can explain these differences: SO is a technical forum for programming languages, while EE is a technical forum for experts and learners in the electronics and electrical engineering domain.

.

Developers' HDL-related questions on the SO and EE forums are dominantly associated with the languages VHDL, Verilog, and System Verilog. We found ten types of HDL-related questions, among which *error* and *review* are more frequent compared to GPPLs, and *code explanation*, *how to test* and *tool search* are new categories. Our results indicate the need to improve the understanding (*review* and *code explanation*), verification (*how to test*) and diagnosis (*error*) of HDL programs.

RQ2: What are the topics in the HDL-related questions?

Motivation

Developers ask and answer HDL-related questions on technical forums such as SO and EE. These questions may represent developers' challenges when learning or creating HDL programs. We use topic modeling to extract meaningful topics from these questions to understand these challenges.

Approach

Extracting and labeling topics. We used the process described in Section 3.2 (Steps 4 and 5) to process the HDL posts, use the topic model (LDA) to extract the topics, and manually label the topic modeling results.

Distribution of topics. Each question contains a distribution of all the resulting topics (each topic is assigned a probability), typically with one or more dominant topics. We assign a topic to a question if the topic has the highest probability in the question. We then calculate the distribution of the topics by measuring the percentage of questions associated with each topic.

Comparing HDL-related topics with GPPL topics. In order to understand the relationship between the topics of HDL-related questions and the topics of GPPL-related questions, we compare our resulting topics with the topics reported in the literature that study the topics of programming-related questions in technical forums, including the topics of programming languages in general [101], mobile programming [51], security-related development [60], client development [58], concurrent programming [59], big data programming [61], Swift pro-

Table 3.3 Topics extracted from HDL related posts on Stack Overflow.

Topic (manual label)	Keywords	Description	% Freq	Similar Topics
Design and implementation	SO: design, implement, code, exampl, synthe, find, vhdl, logic, differ, case EE: design, logic, synthe, implement, exampl, case, differ, thing, code, find	HDL program design and development	SO: 14.5 EE: 12.0 both: 14.0	*OO programming* [58, 62, 101]
Testing and simulation	SO: simul, code, output, work, write, test, problem, time, result, wrong EE: simul, code, error, work, problem, output, write, expect, vhdl, follow	Simulation of the system and checking the results of the test	SO: 11.8 EE: 20.6 both: 13.7	*Testing* [101] *iOS testing* [62] *Unit testing* [58]
Syntax errors	SO: error, code, follow, compil, vhdl, syntax, give, find, expect, problem EE: N/A	Finding errors which occure during compilation	SO: 10.0 EE: N/A both: 7.9	*Compiling* [101] *Build troubleshoot* [63]
Process statements	SO: process, statement, time, simul, event, block, case, signal, wait, variabl EE: process, statement, time, simul, signal, vhdl, block, code, clock, exampl	time and statement of the simulation process	SO: 7.1 EE: 7.4 both: 7.2	N/A
Bit operations	SO: bit, number, result, input, add, sign, output, adder, vhdl, subtract EE: bit, number, output, input, result, shift, width, implement, vector, add	issue concerning bit operations such as numbers, input, and output	SO: 6.6 EE: 9.9 both: 7.1	N/A
File I/O	SO: file, project, find, work, simul, vhdl, read, text, compil, design EE: file, design, find, vhdl, program, project, work, tool, hdl, test	File input and output issues	SO: 6.2 EE: 10.3 both: 7.1	*File operation* [51, 101] *File management* [59] *IO troubleshoot* [63]
Data containers	bit, vector, size, array, index, width, variabl, number, length, concaten EE: N/A	issue concerning vectors and arrays such as index, size and variable	SO:7.9 EE: N/A both: 6.3	*Data structures* [51]
Signal I/O	SO: output, input, port, modul, wire, connect, signal, drive, declar, top EE: output, input, port, connect, wire, modul, warn, drive, signal, pin	Signal input and output matters	SO: 5.7 EE: 6.6 both: 5.9	*Signal troubleshoot* [63]
State machines	SO: state, code, lead, work, display, counter, switch, problem, output, vhdl EE: state, output, input, lead, reset, display, set, high, switch, start	Problems concerning machines states such as display, counter, and switch	SO: 5.6 EE: 7.1 both: 5.9	N/A
Type issues	SO: type, declar, vhdl, function, element, constant, error, std-logic-vector, conver, record EE: type, declar, function, vhdl, error, constant, variabl, vector, defin, size	Issues and errors concerning typing different programming elements	SO: 4.8 EE: 8.0 both: 5.5	*Data types* [62]
Classes and objects	SO: function, call, class, variabl, task, object, systemverilog, declar, pass, system EE: N/A	Issues concerning functions such as calling one, variable declaration and object oriented programming	SO:5.9 EE: N/A both: 4.8	*Value passing & methods* [51] *General programming* [61] *Method call session* [58]
Clocked processes	SO: clock, time, reset, input, high, output, signal, pul, set, clk EE: N/A	Issues concerning the generation of clock, time, and clocked processing	SO: 3.6 EE: 9.0 both: 4.7	N/A
Memory I/O	SO: read, write, memori, address, datum, regist, instruct, ram, store, bit EE: read, write, datum, address, memori, ram, store, regist, block, instruct	Memory and registering issues such as addressing, instructing, reading and writing	SO: 3.1 EE: 5.1 both: 3.5	*Memory/Pointer* [101] *Memory* [60] *Memory consistency* [59] *Memory management* [63]
Data transfer	SO: datum, send, receiv, serial, bit, driver, monitor, sequenc, read, transact EE: datum, bit, send, byte, receiv, serial, start, packet, charact, uart	Issues concerning data transfer including sending, receiving, monitoring and sequencing	SO: 2.3 EE: 3.4 both: 2.5	*Networking* [62, 101] *Serial port communication* [63]
Register operations	SO: regist, shift, bit, warn, latch, constant, block, signal, find, design EE: N/A	Issues concerning register operations	SO: 2.0 EE: N/A both: 1.6	N/A

Our topic modeling results produced 16 topics. As we cannot find a meaning label for one topic, we removed the topic from our analysis.

gramming [62], IoT programming [63], and deep learning development [64]. We identify a similar topic if the definition or keywords of a topic reported in the literature is similar to one of our extracted topics.

Topic popularity. Following earlier work [93] [61] we assessed two metrics for each topic to identify developers' interest in each topic: 1) the median/average view count of the related questions, and 2) the median/average score of the associated questions.

Distribution of topics for each programming language. We first identify the questions associated with each programming language and each topic. Then, for each language, we calculate the distribution of each topic in the questions associated with that language. We focus on the top five languages (VHDL, Verilog, SystemVerilog, HDL, and Chisel), as they account for 98.0% of all HDL-related questions.

Results

We derived 15 topics in the HDL-related questions. While some topics (e.g., *syntax errors*) are common among GPPL (e.g., Java), others (e.g., *bit operations*) are only notable in HDL-related questions. We extracted 16 topics from the HDL-related questions in the SO forum. However, we could not assign a meaningful label to one topic; thus, we removed the topic from our analysis. The removed topic only accounts for 2.1% of SO questions. We also extracted 11 topics from the HDL-related questions in the EE forum, all of which can be mapped to the topics extracted from the SO forum (i.e., the mapped topics share similar keywords). Table 3 shows our final 15 topics extracted from the two forums, their associated keywords, and their frequency in the two forums.. Overall, *design and implementation*, *testing and simulation*, and *syntax errors* are the most important topics in HDL-related questions.

Design and implementation. The most frequently discussed topic is *design and implementation*, which accounts for 14.0% of all questions from both forums. The following is an example question (with over 33K views) associated with the topic of *design and implementation*:

> *I am in the process of writing some Verilog modules for an FPGA design. I looked around the internet to find out how I best parametrize my modules. I see two different methods occurring often. I included an example hereunder of the two different methodologies. Which of these methods is the best way to parametrize modules? What is the difference?*

Testing and simulation is the second most popular topic, accounting for 13.7% of all posts in both forums. Testing is a piece of software that validates that the system performs as expected, and simulation refers to executing a circuit design in a software environment. For instance, one of the most viewed questions associated with this topic is:

> *I have tried this multiple ways, I am a bit desperate now. I have tried to make this clock in my testbench the problem is in simulation it doesn't work or my simulation seems to freeze. I know it has to be the clock.*

Syntax errors is the third most frequently discussed topic, accounting for 7.9% of all postings. A compiler error is a condition that occurs when a compiler fails to build a piece of computer source code, either related to code errors or, more rarely, compiler errors. One of the most viewed questions associated with this topic is:

> *"I'm new to VHDL and am trying to compile the simple exampprovided... I followed the project creation steps in the... bu when I try to compile the project I get the error.*

The topics of *process statements*, *bit operations*, *state machines*, *clocked processes*, and *register operations* are only observed in HDL-related questions. These topics are associated with the particular characteristics of HDLs in contrast to GPPL. For example, one of the most important differences between HDLs and GPPL is that HDLs consider a notion of time [11]. Thus, HDL developers face the challenges of the synchronization of the computing logic (i.e., the topic of *clocked processes*). In addition, in contrast to GPPL, HDLs describe the computing logic at a lower level; thus, low-level operations such as *bit operations* and *register operations* are also challenges faced by HDL developers.

HDL-related questions received similar community attention as questions related to GPPL. Table 4 shows the popularity of the HDL-related questions related to each topic in both forums. In the SO forum, questions related to *type issues*, *data containers*, *syntax errors*, and *bit operation* receive the most attention (i.e., most views) from the community; in the EE forum, questions related to *process statements*, *file I/O*, *signal I/O*, and *type issues*

receive the most attention. According to prior studies on mobile programming [51], big data programming [61], concurrent programming [62], and IoT programming [63], the view counts and scores of these HDL-related questions are similar to those associated with GPPL.

Table 3.4 Popularity of HDL-related topics on both forums.

Topic	Forums	Number of questions	Median View	Average View	Median Score	Average Score
Process statements	SO	934	625	2,472	0.0	1.0
	EE	252	542	**3,024**	1.0	1.2
File I/O	SO	816	681	2,193	1.0	1.1
	EE	**352**	**696**	**2,095**	1.0	**2.8**
Testing and simulation	SO	**1,541**	435	1,575	0.0	0.5
	EE	**702**	448	1780	0.0	0.6
Bit operations	SO	826	722	**3,403**	0.0	0.6
	EE	338	447	2,559	0.0	0.9
Signal I/O	SO	742	579	2,874	0.0	0.7
	EE	227	**550**	2,544	1.0	1.0
Clocked processes	SO	470	572	2,076	0.0	0.6
	EE	307	480	1,938	1.0	1.3
State machines	SO	738	501	1,614	0.0	0.3
	EE	243	462	1,349	1.0	0.8
Memory I/O	SO	412	575	1,570	0.0	0.5
	EE	175	513	1,767	0.0	0.9
Data containers	SO	1,033	**750**	**4,014**	1.0	1.1
	EE	NA	NA	NA	NA	NA
Syntax errors	SO	**1,302**	**743**	2,725	0.0	0.3
	EE	NA	NA	NA	NA	NA
Type issues	SO	636	**793**	**3,131**	1.0	**1.2**
	EE	275	**905**	**3,655**	1.0	**1.5**
Classes and objects	SO	796	621	2,048	1.0	1.0
	EE	NA	NA	NA	NA	NA
Register operations	SO	266	595	2,140	0.0	0.4
	EE	NA	NA	NA	NA	NA
Data transfer	SO	308	451	978	0.0	0.4
	EE	118	498	1,368	1.0	1.0
Design and implementation	SO	**1,891**	610	2,604	1.0	**1.4**
	EE	**410**	445	2,644	1.0	**2.2**

The most important topics are consistent among the programming languages.

Table 3.5 shows the distribution of topics in the questions related to each programming language. As shown in the table, the most critical topics in the two forums, which are *testing and simulation* and *design and implementation*, are consistent among the top five programming languages. In addition, the topics of *syntax errors* and *file I/O* are each among

Table 3.5 Topic distribution (%) in questions related to each language

Topic	Forums	Vhdl	Verilog	System-Verilog	Hdl	Chisel
Process statements	SO	7.8	6.0	5.6	5.7	1.8
	EE	9.0	5.8	8.1	5.4	NA
File I/O	SO	5.9	4.9	5.2	4.4	**11.7**
	EE	**11.6**	9.1	11.9	**19.2**	NA
Testing and simulation	SO	**19.4**	**23.3**	**23.6**	**23.7**	**22.8**
	EE	**21.0**	**20.5**	**19.4**	**17.9**	NA
Bit operations	SO	6.3	6.6	3.3	5.5	2.0
	EE	8.9	**11.2**	11.6	7.5	NA
Signal I/O	SO	3.3	7.3	5.2	5.4	4.5
	EE	4.9	8.1	6.9	5.0	NA
Clocked processes	SO	3.6	3.5	2.4	3.0	1.8
	EE	7.8	10.6	5.2	12.1	NA
State machines	SO	7.0	4.7	1.6	5.0	1.4
	EE	6.9	7.6	4.5	5.0	NA
Memory I/O	SO	2.7	3.1	1.8	3.4	4.3
	EE	5.1	5.6	3.9	2.5	NA
Data containers	SO	5.8	8.0	**10.5**	7.2	7.8
	EE	NA	NA	NA	NA	NA
Syntax errors	SO	**10.2**	**8.8**	5.5	**9.1**	8.8
	EE	NA	NA	NA	NA	NA
Type issues	SO	8.0	1.2	3.1	2.5	3.9
	EE	**10.5**	4.2	**15.2**	7.1	NA
Classes and objects	SO	1.4	4.3	**16.9**	4.6	8.8
	EE	NA	NA	NA	NA	NA
Register operations	SO	1.9	2.1	1.1	1.9	2.2
	EE	NA	NA	NA	NA	NA
Data transfer	SO	2.1	1.5	2.8	1.7	0.6
	EE	4.0	3.1	1.3	1.3	NA
Design and implementation	SO	**13.9**	**13.7**	10.3	**16.0**	**17.0**
	EE	10.4	**14.3**	**12.3**	**17.1**	NA

the most important topics of three languages in at least one of the two forums; *type issues* is among the most significant topics of two languages (VHDL and System Verilog) in the EE forum. Some exceptions exist, for example, *data containers* and *classes and objects* are among the most important topics for System Verilog in the SO forum, while *bit operations* and *type issues* are important topics for Verilog and VHDL in the EE forum, respectively. Overall, our observations indicate that developers of different hardware description languages face similar challenges.

Differences between the topics of questions in SO and EE.

Overall, we observe similar topics of questions in the two forums. We identified 15 topics and 11 topics from the questions in SO and EE, respectively. All 11 topics of the EE questions can be mapped to the topics of SO questions. The four topics that only exist in SO questions include *syntax errors*, *data containers*, *classes and objects*, and *register operations*. The differences can be explained by the fact that SO is a technical forum for programmers, and these four topics are all closely related to programming.

HDL developers share most of the challenges (e.g., syntax errors) with GPPL developers. However, some challenges (e.g., lower level operations such as bit and register operations or synchronization through clocked processes) are only notable in HDLs.
The most important topics in the questions are consistent among different HDLs. Future efforts are needed to help HDL developers address the most important challenges (e.g., *testing and simulation*, *syntax errors*) in HDL development and the challenges specific to HDLs (e.g., *process statements*, *bit operations*).

RQ3:What questions are difficult for developers to get answers?

Motivation

To understand the most challenging areas of developing HDL programs, we analyze the difficulty level of the questions related to different HDLs and topics. Our results can provide insights for future work to support these most challenging areas of HDL development.

Approach

Estimating the difficulty level of each language. We first obtain the set of questions associated with the corresponding tag (e.g., “vhdl”). Then, we use the two heuristics described in Section 3.2 (Step 6) to estimate the difficulty level of the questions associated with the HDL. Specifically, for each HDL, we calculate the percentage of the associated questions

receiving an accepted answer and the median and mean time taken to receive an accepted answer. Similar to RQ2, we focus on the top five HDLs, which account for 98.0% of all the HDL-related questions.

Estimating the difficulty level of each topic. First, we identify the dominant topic for each question (i.e., the topic with the highest probability). Then, for each topic, we use the two heuristics described in Section 3.2 (Step 6) to estimate the difficulty level of the questions associated with the topics. Similarly, for each topic, we calculate the percentage of the associated questions receiving an accepted answer and the median and mean time taken to receive an accepted answer.

Estimating the difficulty level of each topic in each language.

To understand the difficulty level of each topic in each language, following previous approaches, we first identify the questions associated with a certain HDL and a specific topic. Then, we use the two heuristics to estimate the difficulty level of these questions.

Results

HDL-related questions are more difficult than questions related to GPPL to receive accepted answers. Table 3.6 shows the difficulty level of the top five programming languages. Compared to technical forum questions related to GPPL, which present an average ratio of 70.0% questions and a median time of 21 minutes to receive an accepted answer [51], questions related to HDL are less likely and take longer to get accepted answers. Between the two most popular HDLs: VHDL and Verilog, questions related to VHDL are less likely and with a median time of 2:14 hours for SO and 2:09 hours for EE take longer time to receive accepted answers. Moreover, questions related to VHDL (51.8% and 48.8% without accepted answers in SO and EE, respectively) are least likely to get accepted answers. VHDL questions also take the longest to receive accepted answers with 2:09 hours from the EE forum, while Chisel questions with 8:25 hours take the longest to receive accepted answers from the SO forum. We also observe that, in general, questions in the EE forum are more likely (e.g., 51.8% and 48.8% of VHDL questions in SO and EE, respectively, do not have accepted answers) and take a shorter median time (the median time to get accepted answers in SO and EE are 2h19min and 1h44min, respectively) to get accepted answers than SO.

Questions related to *file I/O*, *memory I/O*, *data transfer*, *state machines*, *syntax errors*, and *testing and simulation* are among the most difficult to get community support in the SO and EE forums. Figure 3.2 and Figure 3.3 show the difficulty level of the HDL-related topics in the two studied forums in terms of the two heuristics. We observe

Table 3.6 Difficulty level of questions related to each language in both forums

Language	Vhdl	Verilog	System Verilog	Hdl	Chisel
% questions without accepted answers (SO)	**51.8**	44.8	43.7	42.9	44.4
% questions without accepted answers (EE)	**48.8**	44.9	46.1	45.8	N/A
Median time to get accepted answer (hr:min) (SO)	2:17	1:49	2:28	2:00	**8:25**
Median time to get accepted answer (hr:min) (EE)	**2:09**	1:26	1:46	2:04	N/A
Average time to get accepted answer (hr:min) (SO)	159:31	114:51	128:58	274:19	**464:32**
Average time to get accepted answer (hr:min) (EE)	**168:26**	60:33	44:18	82:10	N/A

that, in the SO forum, questions associated with the topics *syntax errors*, *state machines*, and *testing and simulation* are least likely to get accepted answers, while questions associated with the topics *memory I/O*, *file I/O*, *data transfer*, and *state machines* take the longest time to get accepted answers. In the EE forum, questions related to *memory I/O* and *testing and simulation* are least likely to get accepted answers, while questions related to *data transfer* take the longest time to get accepted answers. These topics represent the most challenging areas of HDL development and indicate opportunities for future work to improve the HDL development practices (e.g., to improve the language abstraction of handling file/memory I/Os, to provide actionable information to address syntax errors, or to develop tools to improve testing and simulation). In particular, **the topics of *syntax errors* and *testing and simulation* are among both the most important challenges (RQ2) and the most difficult challenges.**

The most difficult topics are consistent among different programming languages. Tables 3.7 and 3.8 show the difficult aspects of the questions associated with each language and each topic in SO, and Tables 3.9 and 3.10 show that for questions in EE. In general, the most difficult topics are consistent across the different languages. For example, the topics of *state machines* and *syntax errors* in SO and the topics of *memory I/O* and *data transfer* in EE consistently have the highest ratio of questions without accepted answers (only with few exceptions). Similarly, the topics of *file I/O*, *memory I/O* and *state machines* in SO and the topics of *memory I/O* and *design and implementation* in EE consistently take the longest time to get accepted answers (with few exceptions). Our results indicate that the

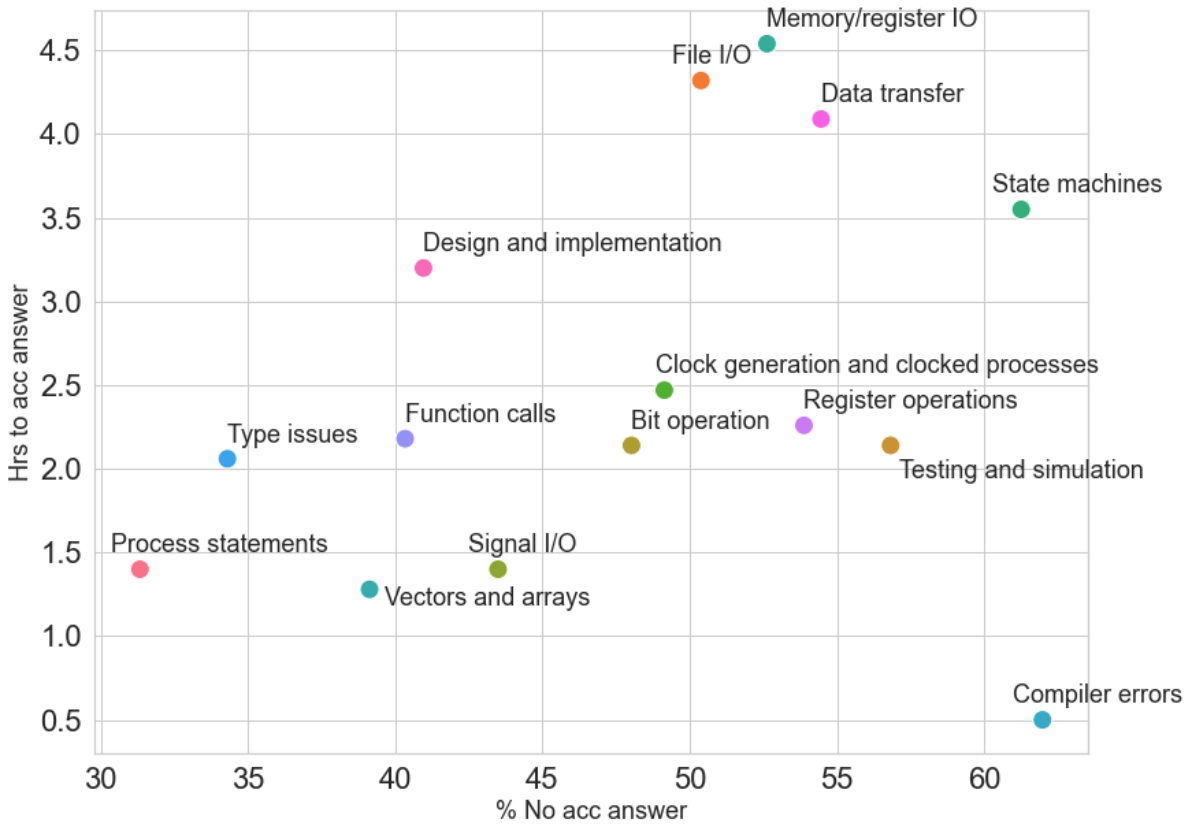

Figure 3.2 The difficulty aspect of HDL-related topics on SO.

most challenging areas of HDL development are consistent across different HDLs.

Differences between difficult questions in SO and EE. We observe that, in general, the types of difficult questions in terms of programming languages and topics are similar in the two studied forums. We also observe that, in general, questions in the EE forum are more likely and take a shorter average time to get accepted answers, except for VHDL-related questions. The difference may be explained by the fact that the EE forum is more focused on hardware (i.e., with users more focused on HDLs), while SO covers both GPPLs and HDLs and has more diverse users.

In general, HDL-related questions are less likely and take longer than questions related to GPPL to get accepted answers. The most difficult topics (*file/memory I/O*, *data transfer*, *state machines*, *syntax errors*, *testing and simulation*) are consistent across different languages. Future efforts are needed to develop better resources and tools to support these most challenging areas, such as improving the language abstraction for handling file/memory I/Os, providing actionable information to address syntax errors, or developing tools to improve testing and simulation.

Table 3.7 The percentage of questions without accepted answers in SO forum (per topic and language)

Topic	Vhdl	Verilog	System-Verilog	Hdl	Chisel
Process statements	31.3	28.7	29.7	20.0	**66.7**
File I/O	56.7	45.7	43.4	46.2	38.6
Testing and simulation	**65.0**	**52.6**	51.4	**58.1**	45.2
Bit operations	55.3	40.8	43.3	30.6	40.0
Signal I/O	52.8	40.2	38.4	35.4	45.5
Clocked processes	50.2	46.6	54.3	37.0	44.4
State machines	**66.8**	**53.6**	**61.7**	50.0	**71.4**
Memory I/O	57.2	50.8	47.3	46.7	47.6
Data containers	44.5	35.1	34.0	29.7	42.1
Syntax errors	64.2	**56.9**	**64.2**	**53.8**	**67.4**
Type issues	33.9	31.0	34.8	22.7	36.8
Classes and objects	57.6	41.8	36.9	46.3	41.9
Register operations	59.1	50.0	**54.5**	52.9	18.2
Data transfer	**66.9**	49.5	41.5	**53.3**	33.3
Design and implementation	40.2	39.6	46.0	39.2	41.0

Table 3.8 The median time (hr:min) to get accepted answers in SO forum (per topic and language)

Topic	Vhdl	Verilog	System-verilog	hdl	Chisel
Process statements	1:48	1:26	2:20	1:10	9:33
File I/O	**4:16**	**3:43**	**4:54**	**3:16**	7:12
Testing and simulation	2:44	1:30	2:53	1:31	9:45
Bit operations	2:33	2:03	1:35	2:36	12:31
Signal I/O	1:34	1:25	2:04	2:09	7:14
Clocked processes	1:53	2:47	**4:54**	1:25	15:21
State machines	**4:27**	3:05	2:57	**4:16**	**86:47**
Memory I/O	**4:31**	**4:27**	4:17	**5:27**	**14:25**
Data containers	1:22	1:12	1:20	2:06	6:02
Syntax errors	0:51	0:42	1:19	1:23	4:31
Type issues	2:06	1:31	2:30	2:34	7:07
Classes and objects	1:08	2:16	2:12	1:30	9:56
Register operations	1:37	1:59	3:30	0:26	**15:06**
Data transfer	3:24	**3:31**	**6:14**	1:08	1:54
Design and implementation	3:29	2:57	3:41	2:35	10:41

Table 3.9 The percentage of questions in EE forum without accepted answers (per topic and language)

Topic	**Vhdl**	**Verilog**	**System-Verilog**	**Hdl**
Memory I/O	**53.5**	**63.3**	50.0	**66.7**
type issue	40.8	29.9	21.3	23.5
process statement	36.6	38.3	48.0	53.8
Design and implementation	40.3	34.6	**60.5**	43.9
Testing and simulation	**64.1**	56.8	53.3	**55.8**
Data tarnsfer	**52.9**	**42.0**	**100.0**	33.3
State machines	52.5	48.8	**57.1**	41.7
Bit operations	51.0	45.3	33.3	**55.6**
clocked processes	36.8	31.0	50.0	31.0
File I/O	46.7	**53.1**	56.8	47.8
Signal I/O	48.2	39.7	33.3	50.0

Table 3.10 The median time (hr:min) to get accepted answers in EE forum (per topic and language)

Topic	**Vhdl**	**Verilog**	**System-verilog**	**hdl**
Memory I/O	**3:55**	**2:30**	0:46	**217:27**
type issue	1:33	1:34	1:46	2:38
process statement	1:10	1:09	1:53	1:14
Design and implementation	**3:08**	1:06	2:29	**4:45**
Testing and simulation	2:44	**1:51**	**2:44**	1:38
Data tarnsfer	**5:02**	**1:53**	N/A	1:05
State machine	2:47	1:16	**6:49**	1:18
Bit operations	1:40	0:58	0:46	0:35
Clocked processes	1:29	1:39	1:02	**3:36**
File I/O	2:47	1:26	**5:51**	1:53
Signal I/O	1:36	1:14	1:16	2:22

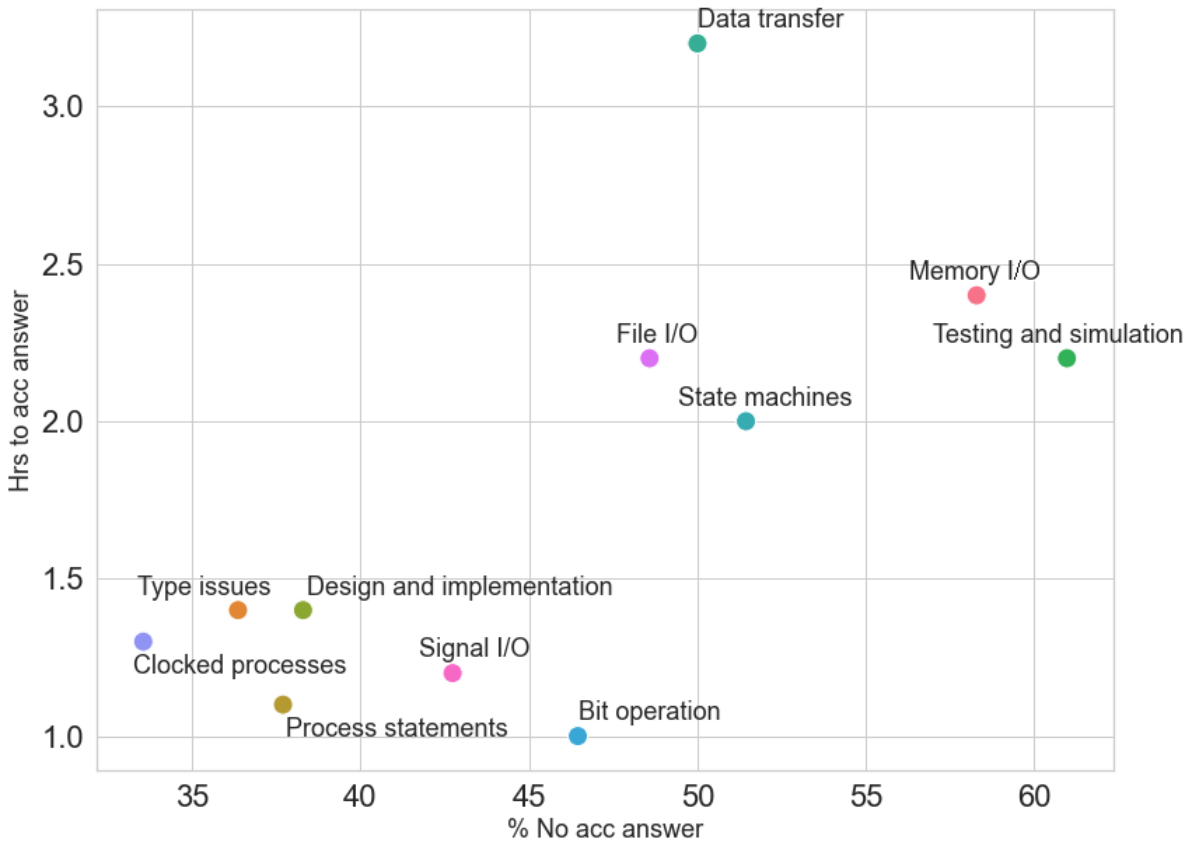

Figure 3.3 The difficulty aspect of HDL-related topics on the EE forum.

3.4 Discussions and Implications

Comparing technical questions related to HDLs and GPPLs. Computer architectures are at the point of shifting from a general-purpose approach to a hardware/software co-design approach (i.e., domain specific architecture) [2]. To efficiently develop such domain specific architectures, we need balanced support for developing programs in GPPLs and HDLs. In our study, we observe that HDL developers ask new types of questions related to *code explanation*, *how to test*, and *tool search*, which indicate the lack of resources and tools to assist HDL developers in developing, understanding, and testing HDL programs. We also observe that HDL-related questions include new topics such as *bit operations* and *register operations* that are related to low-level programming, which indicate the need for more abstract constructs and design patterns to improve the efficiency of HDL program development. Finally, we observe that HDL-related questions are less likely and take a longer time to get accepted answers in technical forums than GPPL-related questions, which may be explained by the difficulty of HDL-related questions or the smaller size of the active HDL community in the technical forums. **The software engineering community should leverage its expe-**

rience in advancing GPPL development to help in advancing HDL development languages and methodology, providing HDL development tools and resources, and promoting community support in technical forums.

Comparing HDL-related questions in SO and EE. Unlike traditional GPPL program development, HDL program development typically needs knowledge in programming and circuit design. The SO forum has a large user base experienced in programming, while the EE forum provides a user base with expertise in circuit design. On the one hand, we observe that, overall, HDL developers ask a wider range of questions in SO than in EE. However, they are more likely to ask background and conceptual questions in EE and programming specific questions on SO. This difference can be explained by the fact that SO is a general programming language forum, while EE is a forum for hardware practitioners and learners to exchange hardware-related questions. On the other hand, we also observe that, in general, questions in the EE forum are more likely to get accepted answers and faster, which may indicate that HDL-related questions are relatively difficult for general SO users while being easier for hardware practitioners and learners in the EE forum. **Stack Exchange and its moderators can leverage the community size and expertise in both forums to collectively recommend experts to answer questions related to HDLs.**

3.5 Threats to Validity

External validity. This work analyzes HDL-related questions on technical forums (SO and EE) to understand the challenges of developing HDL programs. However, HDL developers may also communicate their discussions in other forums or media (e.g., Code Review forums and Google groups). Nevertheless, after searching available forums related to HDLs, we found that the most recent posts related to HDL questions on the internet are available on the studied SO and EE forums, while other forums with information related to HDL languages are not as active. Future work considering other data sources may complement our study. In addition, our identified and collected questions may not cover all the questions on Stack Overflow. To alleviate this threat, we follow prior work [102] and use an iterative method to identify the relevant tags.

Internal validity. We use topic modeling (LDA) to cluster the Stack Overflow questions based on the intuition that the same clusters would have similar textual information. However, different clusters of questions may exist when a different approach is used. To ensure the quality of the clusters, we manually reviewed the resulting topics and assigned meaningful labels to them.

Construct validity. In RQ1, we manually analyze the categories of HDL-related questions

on Stack Overflow. Our results may be subjective and depend on the researchers judgment who conducted the manual analysis. The study's three authors collectively conducted the manual analysis and reached a consensus for each question to mitigate the bias. In RQ2 and RQ3, the hyperparameters of the topic models (e.g., the number of topics K) may impact our findings. We tuned the hyperparameters and used the topic coherence score to select the most suitable parameters to mitigate this threat. In addition, in RQ3 maybe because of the smaller size of HDL community, HDL-related questions are less likely and take longer than questions related to GPPLs to get accepted answers.

3.6 Summary

This study considers the challenges that developers of HDLs face by examining HDL-related technical questions in the SO and EE forums. We observe that developing programs in HDL faces both similar challenges as GPPL and challenges specific to HDLs (e.g., lower level operations such as bit and register operations or synchronization through clocked processes). We also observe that it is more challenging for developers of HDL programs to receive community support than GPPL (i.e., HDL-related questions are less likely and take longer to get accepted answers) which calls for efforts to enhance developer engagement or platform support in answering questions in technical forums (e.g., SO). In addition, our work identified opportunities for the research community to improve the practices of developing HDL programs, such as to improve the language abstraction for handling bit/register operations or file/memory I/Os, to provide actionable information to address syntax errors, and–or to develop tools to improve testing and simulation.

CHAPTER 4 STUDYING DNN HARDWARE ACCELERATION PRACTICES IN GITHUB PROJECTS

Context: Computer architectures are at the point of shifting from a general-purpose approach to a hardware-software co-design approach (i.e., domain specific architectures). In recent years, hardware acceleration for deep neural networks (DNNs) has become a popular application area of domain-specific hardware-software co-design.

Objectives: We aim to understand the practices and challenges of hardware description language (HDL) development in real-world projects by examining open-source HDL-related projects in the context of DNN hardware acceleration.

Method: We examine 321 public GitHub projects related to hardware acceleration for DNNs, to understand the practices and challenges of developing HDL code in real-world projects. We study the categories of projects that leverage hardware acceleration for DNN, the distribution of programming languages, developer profiles of these projects, and the purposes of using hardware acceleration for DNN in these projects.

Results: We observe that DNN developers leverage hardware acceleration for a variety of DNN models (e.g., CNN and RNN). We also observe that HDLs (e.g., VHDL) contribute to a higher portion of code than general-purpose-programming-languages (GPPLs) (e.g., C) in the studied DNN acceleration projects. However, there are fewer developers working on HDLs than those working on GPPLs. Finally, we observe that most of the projects leverage hardware acceleration for DNN inference while only a small portion of them leverage hardware acceleration for training DNN models.

Conclusion: Hardware-software co-design is increasingly popular in applications such as DNN acceleration. However, there is an imbalance between HDL and GPPL development (e.g., imbalance between HDL and GPPL developers). Our community should provide more empathises to support HDL program development, to avoid the HDL-related components being the bottlenecks of hardware-software co-design.

4.1 Introduction

Deep Neural Networks (DNNs), also known as Deep Learning, are a subset of AI, which, according to John McCarthy, the computer scientist who invented the phrase in the 1950s, is "the science and engineering of constructing intelligent computers that can achieve goals as people do". DNNs are the cutting-edge solutions for various applications, including computer

vision, speech recognition, and natural language processing, among others. Artificial Neural Networks (ANNs) are a mathematical architecture that connects a vast number of essential elements known as neurons, each of which can make simple mathematical decisions [9]. An external neural network has only three layers: the input layer, one hidden layer, and the output layer. As the number of hidden layers in a neural network grows, it becomes a Deep Neural Network (DNN). As a result, Deep Learning can be thought of as a subset of Artificial Neural Networks with many processing layers. They are more precise and continue to improve as more neuron layers are added. Feed-Forward Neural Network, Recurrent Neural Network (RNN), and Convolutional Neural Network (CNN) are three popular Deep Neural Network models [9].

However, DNNs' increased accuracy comes at the cost of significant computational complexity. A DNN model is typically very complex and takes a long time and expensive resources to perform training and inferences. While general-purpose compute engines, particularly graphics processing units (GPUs), have been the basis for much DNN processing, there is growing interest in more specialized DNN acceleration. Researchers and practitioners have proposed various approaches to make DNN model training and inferences more efficient. These approaches typically fall into two categories: 1) model optimization (the software way) and 2) hardware acceleration (the hardware way). In particular, hardware acceleration for DNN models has drawn more and more attention [10, 71, 88, 103]. There has been recent evidence from the field of deep learning that FPGAs (Field-Programmable Gate Arrays) is an essential factor in the acceleration of DNNs (Deep Neural Networks) [88]. AFTER MANUAL TRANSFORMATION, an HDL (Hardware Description Language) vendor tool is used to synthesize DNN from a high-level language such as Python. HDL expertise is required for this transition, limiting the use of FPGAs [10]. In data centers, FPGAs are now widely used to offload GPU- and CPU-based inference engines [9]. We are in the early stages of defining, expanding, and deploying such capabilities, starting with targeted FPGAs, model development and optimization frameworks, and an ecosystem of supported libraries. Over the next five years, FPGA capabilities are expected to improve rapidly, allowing them to tackle a wide range of real-world applications [9]. Despite the benefits and rapid growth of hardware acceleration for DNNs, no work studies the practices of using hardware acceleration for DNNs in real-world projects. In this research, we investigate 321 open-source GitHub projects that use hardware acceleration for DNN models. Our research aims to answer the following research questions (RQs):

RQ1: What are the categories of the projects that leverage hardware acceleration for DNN? To understand the categories of the projects, we manually analyzed the 321

projects and examined the types of DNN models and the associated tasks that leverage hardware acceleration. DNN developers leverage hardware acceleration for a variety of DNN models (e.g., CNN and RNN). The studied projects mainly leverage hardware acceleration for CNN models, for the tasks such as image classification and object detection. On the other hand, hardware acceleration is used much less frequently for RNNs.

RQ2: What are the distributions of programming languages and developer profiles in the projects leveraging hardware acceleration for DNN? We extracted the programming languages used in the DNN hardware acceleration projects and analyzed the distributions of HDL and GPPL languages in these projects. Further, we investigated the profiles of the developers of these projects to understand the differences between the developers of HDLs and GPPLs. HDLs (e.g., VHDL) contribute to a higher portion of code than GPPLs (e.g., C) in the studied DNN acceleration projects. However, there are fewer developers working on HDLs than those working on GPPLs. Our observation indicates the imbalance between HDL developers and GPPLs developers. Researchers are the most popular category of developers who work on the DNN acceleration projects, while the developers contributing to GPPL code have more diverse profiles than those contributing to HDL code.

RQ3: What are the purposes of using hardware acceleration in the projects leveraging hardware acceleration for DNN? We manually analyzed the 321 DNN hardware acceleration projects to identify the purposes of using hardware acceleration for DNNs in these projects. Most of the projects leverage hardware acceleration for DNN inference while only a small portion of them leverage hardware acceleration for training DNN models. In addition, a small portion of projects use hardware acceleration for benchmarking purposes. Our results can help practitioners and researchers understand the practices of DNN hardware acceleration in real-world projects. Furthermore, the results can help to understand the practices and challenges of HDL code development and software-hardware co-design in the DNN acceleration context.

Chapter organization. The rest of the chapter is organized as follows. In Section 4.2 we describe the experiment setup of our study. In Section 4.3, we present and discuss our results for answering our research questions. In Section 4.4, we discuss the implications of our findings. Section 4.5 discusses threats to the validity of our findings. Finally, Section 4.6 gives the summary of the study.

4.2 Experiment Setup

This section describes the design of our study.

4.2.1 Overview of the Study

This study aims to examine open-source projects that are related to hardware acceleration for DNNs on GitHub [1] to understand the practices and challenges of DNN hardware acceleration and HDL development in real-world projects. We study the categories of projects that leverage hardware acceleration for DNN, the distribution of programming languages, developer profiles of these projects, and the purposes of using hardware acceleration for DNN in these projects. In order to filter the DNN hardware acceleration projects, we initially used a series of heuristic criteria and applied them to the repositories on GitHub. Then, we manually verified each of the resulting repositories to remove false positives. In RQ1, we manually analyzed the categories of the projects that leverage hardware acceleration for DNN. In RQ2, we automatically extracted the distributions of the programming languages used in hardware acceleration for DNN projects. In addition, we manually extract the profile of developers who participate in these projects. Finally, in RQ3, we manually analyzed the purpose of using hardware acceleration for DNN projects. In the rest of the section, an in-depth description of the details of our data collection procedure and analysis approaches is provided.

4.2.2 Data extraction and analyses steps

First, we extracted a database of related projects listed on GitHub as of January 2022. Then, we took the three following steps to carefully extract the projects entitled Hardware Acceleration for DNN. Figure 4.1 shows an overview of our data extraction and analysis process. In this study we focus on FPGA-based acceleration projects because FPGA-based accelerators are more popular than GPU and ASIC-based accelerators due to their high performance, customizable and flexible design, high throughput with large parallelism, faster data transfer, and high speed of a DNN implemented on them [9].

Two criteria were used to find projects related to hardware acceleration for DNN: First, to better understand the nature of the project, its description must be provided in English. Second, the project must be listed in a mainline repository, not in a fork or other repositories.

Step 1: **Search candidate projects (first round)**.
We filtered the search results and identified the DNN acceleration related projects. First, authors identified a set of keywords related to DNN acceleration, then they checked the relation of the keywords with the DNN acceleration projects. Finlay ten initial keywords selected, and added to the keyword "FPGA". Our initial set of keywords includes "*FPGA deep neural network*" , "*FPGA*" AND "accelerator", "*FPGA*" AND "neural network" , "*FPGA*" AND

[1] https://github.com

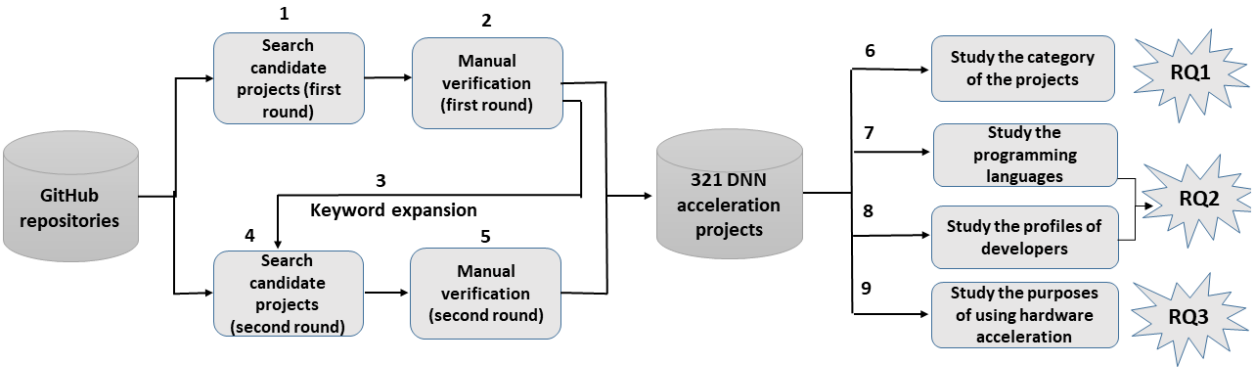

Figure 4.1 An overview of our data extraction and analysis process.

"network", "*FPGA*" AND "learning", "*FPGA*" AND "tensor flow", "*FPGA*" AND "keras", "*FPGA*" AND "pytorch", "*FPGA*" AND "TVM" and "*FPGA*"AND "caffe".

Step 2: **Manual verification (first round).**

We ended up with 647 projects and extracted topics from the 647 projects and manually analyzed extracted topics to find the related ones to hardware acceleration for DNN.

Step 3: **Keyword expansion.**

We selected 15 topics, including *"FPGA" AND "CNN", "hardware" AND "acceleration", "neural network" AND "Xilinx", "FPGA" AND "high-level synthesis", "FPGA" AND "neural network vivado", "Altera" AND "FPGA" AND "neural network", "TensorFlow" AND "FPGA", "OpenCL" AND "FPGA" AND "neural network", "FPGA" AND "convolutional neural networks" ,"pynq" AND "FPGA", "Verilog" AND "FPGA" AND "neural network", "PyTorch" AND "FPGA", "quantization" AND "acceleration", "FPGA" AND "accelerators"* and "*FPGA" AND "neural networks*".

Step 4: **Search candidate projects (second round)**.

By using a new set of keywords in topic search and UI search, we extracted 1017 projects from GitHub.

Step5: **Manual verification (second round)**.

To improve the quality of the selected projects, the projects' descriptions were manually inspected, and those not closely relevant to hardware acceleration for DNN, like documentation for hosting DNN acceleration and lecture notes, were removed. Finally, we arrived at 321 projects directly linked to hardware acceleration for DNN applications as final projects. For the 321 selected projects, we used the GitHub Rest API [2] to obtain all relevant infor-

[2]https://docs.github.com/en/rest

mation, including information on projects title, README file, the number of programming languages, and the number of developers that participate in the projects. We analyze our collected GitHub projects information to answer our research questions through the following steps.

Step 6: Study the category of the projects.
In this step, we carefully reviewed the description and information linked with the projects to determine the category of DNN models that leverage hardware acceleration for DNN projects (e.g., CNN). In addition, to identify the type of tasks in DNN projects, we manually examined and classified the tasks applied in DNN hardware acceleration projects. The resulting step is used to answer our RQ1.

Step 7: Study the programming languages
In this step, we extracted the numbers and distributions of programming languages involved in each of the 321 projects by using the GitHub Rest API. In particular, we compared the distribution of HDL and GPPL programming languages in these projects.

Step 8: Study the profile of developers.
In this step, we extracted the number of developers involved in three groups of programming languages GPPL, HDL, and both GPPL-HDL) by using the GitHub Rest API. To classify the developers, we studied the profiles of 72 developers who had contributed to HDL code and a random sample of 72 developers who had only contributed to GPPL code. We extracted their name, email address, and GitHub profile, followed by a manual investigation of their LinkedIn profile, blog, and any information linked to developers from the web to identify their background and area of work. Based on their background and current job we defined the category and classified them. The results of steps 4 and 5 are used to answer our RQ2.

Step 9: Study the purpose of using hardware acceleration.
In this step, we manually examined the description and code of the projects to identify developers' purposes of leveraging hardware acceleration for DNN models (e.g., for model training or model inference).

4.3 Experiment Results

In this section, we report and discuss the results of our three research questions. We first present the motivation and approach for each research question, then discuss the results for answering the research question.

RQ1: What are the categories of the projects that leverage hardware acceleration for DNN?

Motivation

In order to understand the practices of using hardware acceleration in real-world DNN projects, we first study the characteristics of these projects, including the types of DNN models used in these projects and the associated high-level tasks that these DNN models are used for. Our results can provide a high-level understanding of the current status of hardware acceleration in the wild in real-world DNN projects.

Approach

Identify the category of the model. We manually analyzed the README file of the projects and also examined codes and articles which are linked in the projects to identify the type of DNN model used in them. Find more details of our manual analyses through the following steps.
Step 1. One researcher independently examined the description, codes and papers available in the projects to identify the type of the models.
Step 2. Another researcher reviewed and discussed with the first researcher about type of the model that leverage hardware acceleration for DNN projects and check the labeling.
Step 3. Based on the common understanding, the first researcher revisited the labels and defined the final category of the models.

Identify the category of the tasks. 321 projects were manually inspected to understand the type of tasks in the projects that leverage hardware acceleration for better performance. We followed a similar process as for identifying the category of models to identify the categories of the tasks.

Results

DNN developers leverage hardware acceleration for a variety of DNN models (e.g., CNN and RNN), while CNN dominates the usage scenarios. Table 4.1 shows our identified DNN models from the studied projects that leverage hardware acceleration. The result shows that CNN uses hardware acceleration most frequently (accounting for 56.0% of the cases) in the studied projects. In comparison, although developers also use hardware acceleration for RNN models, it only accounts for 1.0% of the cases. Below, we describe each type of models that leverage hardware acceleration in the studied projects.

CNN (Convolution Neural Networks) is a form of artificial neural network used for image/object identification and classification in Deep Learning. CNN is the most frequent models using hardware acceleration since it is typically very computationally expensive. PipeCNN [104] is an example of an OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks (CNNs) which aims to provide a generic, yet efficient OpenCL-based design of CNN accelerator on FPGAs, and achies improved throughput in inference computation. The second most popular model applied in the hardware acceleration projects for DNN is **General Deep Neural** Network model with 29.9 %. We considered this category since we could not allocate some projects to any specific model, and put them under General Deep Neural Network. SpooNN [105] is an example of general DNN implementation that enable an end-to-end capability to perform inference on FPGAs; starting from training scripts using Tensorflow to deployment on hardware.

The next popular DNN models are **BNN** with 2.1% and SNN with 1.8%. An example of BNN Accelerator is developed in Google brain project [106] which performed a Go implementation and corresponding APIs for acceleration of Binarized Neural Network (BNN) on FPGAs. Gyro [107] is an example of FPGA-based Spiking Neural Network accelerator. As the results show, **RNN** with 1.0% has the least popular among the DNN models, which is used in DNN hardware acceleration projects.

Table 4.1 The categories of DNN models that use hardware acceleration in the studied projects.

Type of model	**% Frequency**
CNN (Convolution Neural Network)	**56.0**
General DNN (Deep Neural Networks)	37.0
BNN (Binarized Neural Network)	2.1
SNN (Spiking Neural Network)	1.8
RNN (Recurrent Neural Network)	1.0
Unknown	2.1

The studied projects dominantly leverage hardware acceleration for general purposes or image Classification-related tasks. Table 4.2 shows the type of task that leverages hardware acceleration for DNN projects. 57.6% of the tasks of the projects are related to **General purpose**. General purpose is the process related to general CNN models. We could not determine the specific task in this category but the projects of this category leverage hardware acceleration during the process. The second prevalent task to leverage hardware acceleration for DNN projects is image classification.

Image Classification is the process of enhancing or extracting valuable information from an image by performing various operations on it. For this form of signal processing, the input

Table 4.2 The categories of high-level tasks that use hardware acceleration in the studied projects.

Task	**% Frequency**
General purpose	57.0
Image classification	26.4
Object detection	7.1
Text classification	2.4
Audio and video recognition	1.5
Unknown	5.6

is an image, and the output can either be that same picture or some of its characteristics or attributes the following is example in image classification: Image Classification using CNN on FPGA [108] is an example of designing a Trained Neural n/w (CIFAR-10 dataset) on FPGA to classify an Image I/P using deep-learning concept(CNN- Convolutional Neural Network). Another task that leverages hardware acceleration for the DNN projects is Object detection with 7.1%. **Object Detection** is an area of computer vision and image processing that focuses on identifying specific types of objects (such as people, buildings, and cars) in digital photos and videos. the following is example in object detection: Inferno [109] is an example of this category of tasks, which is a FPGA Deployable Fire Detection Model for Real-Time Video Surveillance Systems Using Convolutional Neural Networks.

Text Classification is a category of algorithms for processing text data in big scales. Hardware accelerated text classification allocate for 2.4% of tasks.

Audio and Video recognition is a capability which enables a program to process human speech into a written format. Here we have an example for repository in audio and video recognition: ECE 5775 [110] is a neural network-based method for recognizing speech commands with fixed-latency on a Xilinx Zedboard. For the **Unknown** category, there is no more information to identify tasks.

DNN developers leverage hardware acceleration for a variety of DNN models (e.g., CNN and RNN). The studied projects mainly leverage hardware acceleration for CNN models, for the tasks such as image classification and object detection. On the other hand, hardware acceleration is used much less frequently for RNNs.

RQ2: What are the distributions of programming languages and developer profiles in the projects leveraging hardware acceleration for DNN?

Motivation

To understand the practices and challenges of hardware-software co-design in the DNN acceleration context, we analyze the distribution of programming languages used in the studied projects and the profiles of the developers contributing to the code written in these programming languages. Our results can provide insights for improving such hardware-software co-design practices in the DNN hardware acceleration context and beyond.

Approach

In this RQ, we study the programming languages used in the studied DNN acceleration projects and the characteristics of the developers contributing to these projects.

Analyzing the distribution of programming languages. The DNN hardware acceleration projects employed 114 programming languages, which we extracted using REST API. In addition, we extracted the line of codes for each programming language and calculated the total written lines of code for each languages in 321 projects. Based on the percentage of written code in each language and percentage of participating a language in the projects, we keep those languages where the average proportion of codes in all projects is more than 1.0%. Even though the percentage of code for Python and Jupyter Notebook is 0.2% and 0.6%, respectively, their contribution to projects is more significant than 40%, and the authors decided to keep them in the languages category. We calculate both the median and average portion of contributions of each language in the projects.

Analyzing the profile of developers. By using the GitHub API we first extracted the total number of developers. Then we extracted those developers who made changes to the considered top 11 programming languages we identify previously. Finally, we check our final data of developers and remove repeated names and ides and extract developers information including developers' names, email addresses, and GitHub profiles URL links. We manually searched their LinkedIn profiles by using their GitHub profile information or googling their email address and name and tried to find developers information from the web. Based on the programming languages that developers work with, the authors classify the developers in three groups, including:

- GPPL: the developers of this group **only** work on GPPL programming language group
- HDL: the developers of this group **only** work on HDL programming language group

- Both GPPL-HDL groups: the developers of this group only work on at **least one** language from HDL group and at **least one** language from GPPL group

The authors manually analyzed their current information and background and classified them into the categories we obtained from our analyses. We have following steps to manual labeling of the developers: Step 1. One researcher independently examined the description of GitHub profile, work experience and Linkdin profile of developers to identify the category of the developers. Step 2. Another researcher reviewed and discussed with the first researcher about the category of the developers and check the labeling. Step 3. Based on the agreement, the first researcher revisited the labels and defined the final category for developers.

Result

Developers use a wide range of programming languages in the DNN acceleration projects, including HDLs (VHDL, Verilog, System-verilog), and GPPLs (Ada, C, C++, Objective-C, Coq, Tcl, Jupyter notebook, Python). According to Table 4.4, the most popular programming language in the studied DNN hardware acceleration projects is **VHDL** on average accounting for 41.8% of the code in the studied projects. **ADA** is the most commonly used language in the GPPL group, accounting for 10.9% of the code in the projects.

Table 4.3 The distributions of programming languages code involved in the FPGA-related projects.

Language	VHDL*	Ada	Verilog*	C	C++	Objective -C	Coq	System* verilog	Tcl	Jupyter notebook	Python
Average percentage	41.8	14.1	12.9	10.9	7.1	2.5	1.7	1.3	1.0	0.6	0.2
Median percentage	33.3	0.0	4.4	2.5	1.0	0.0	0.0	0.0	0.2	0.0	0.0
Percentage of projects	28.6	2.4	38.0	44.8	41.4	9.0	8.4	17.4	30.2	16.5	45.7

The * sign indicates HDL programming languages

Overall, HDLs contribute to a bigger portion of code in the DNN acceleration projects than GPPLs. Although the number of programming languages used in the projects in the GPPL group is more significant than the number of programming languages used in the HDL group, HDLs contribute to more code than GPPLs. In fact, HDL code contributes to 56.1% of the code in the studied projects on average, while GPPL code contributes to 38.4% of the code on average.

More developers contribute to GPPL code than those contribute to HDL code in the studied DNN acceleration projects. The distributions of developers involved

in each programming language are shown in table 4.4. It should be mentioned that several developers work with multiple programming languages. The results indicate that Python with 42.8% is the most popular programming language among developers. Despite having the highest percentage of codes in the projects, VHDL with 1.2% has the fewest developers Table 4.5 shows that 55.5% of developers only contribute to GPPL code, while only 1.2% of developers only contribute to HDL code; 16.3% of developers contribute to both GPPL and HDL code.

Researchers are the most popular category of developers who work on the DNN acceleration projects, while the developers working on GPPLs in the DNN acceleration projects have more diverse profiles than the developers working on HDLs. We classified developers' profiles into eight categories, including researcher, software engineer, student, AI/ML engineer, data scientist and data engineer, professor, hardware acceleration developer and hardware engineer. As shown in table 4.6, Developers with all categories of profiles contribute to GPPL code. Researchers, software engineers, students, AI/ML engineers, and hardware engineers contribute to both GPPL code and HDL code. Only students, professors, and hardware acceleration developers The result shows that in GPPL-only and both GPPL-HDL groups, the highest percentage of developers are in the **researcher** category. Software engineer is the second popular profile that contribute to GPPL code, while hardware engineer is the second popular profile that contribute to both GPPL and HDL code. 40% of the developers of the HDL-only group are **Students** and most of their contributions in the projects are related to university projects. The other developers include AI/ML engineers with 8.3% in GPPL-only and 3.0% in both GPPL-HDL groups.

HDLs (e.g., VHDL) contribute to a higher portion of code than GPPLs (e.g., C) in the studied DNN acceleration projects. However, there are fewer developers working on HDLs than those working on GPPLs. Our observation indicates the imbalance between HDL developers and GPPLs developers. Researchers are the most popular category of developers who work on the DNN acceleration projects, while the developers contributing to GPPL code have more diverse profiles than those contributing to HDL code.

RQ3: What are the purposes of using hardware acceleration in the projects leveraging hardware acceleration for DNN?

Motivation

DNN projects can leverage hardware acceleration for different purposes (e.g., model inference). However, the purposes of using hardware acceleration in real-world DNN projects are

Table 4.4 The distributions of developers involved in the DNN acceleration projects according to their involved programming languages.

Languge	Number of contributers	% Of contributors
Python	176	42.8
C	100	24.3
C++	74	18.0
Coq	62	15.1
Verilog	62	15.1
Tcl	50	12.2
Jupyter Notebook	50	12.2
Objective-C	23	5.6
SystemVerilog	22	5.4
VHDL	5	1.2
Ada	0	0.0

Table 4.5 The distributions of developers involved in the FPGA-related projects.

Languge	Number of contributers	% Of developers
GPPL group	228	55.5
Both HDL - GPPL	67	16.3
HDL group	5	1.2

Table 4.6 classification of developers in the FPGA acceleration for DNN projects

Classification of contributors	%GPPL-only	%HDL-only	%Both GPPL-HDL
Researcher	25.0	N/A	16.4
Software engineer	13.1	N/A	13.4
Student	11.1	40	13.4
AI/ML engineer	8.3	N/A	3.0
Data scientist and data engineer	6.9	N/A	N/A
Professor	2.7	20	N/A
Hardware acceleration developer	2.7	20	N/A
Hardware engineer	1.3	N/A	13.4
Unknown	28.8	20	39.9

not clear. Thus, in this RQ, we manually studied the DNN acceleration projects to understand developers' purposes of using hardware acceleration for their DNN models. Our results can provide insights for future work to support or improve hardware acceleration to these purposes.

Approach

Follow the previous approach used in (RQ1 and RQ2) we manually analyzed the description of the projects and also examined code and papers in projects related to FPGA acceleration for DNN when needed to identify the purpose of using hardware acceleration. In our manual analyses one researcher independently examined the description, codes and papers available in the projects to identify phase of the models which is used for hardware acceleration. Another researcher reviewed and discussed with the first researcher about model phases that leverage hardware acceleration for DNN projects and check the labeling. Based on the common understanding, the first researcher revisited the labels and defined the final category of the phase of model. We defined our categories in four phases, including training of the model, inferencing, tuning of a model, and benchmarking of the model. Based on hardware acceleration usage for DNN projects, we categorized them to determine in which part of the project processing the benefits of hardware acceleration were used.

Table 4.7 The phase of models leveraging hardware a acceleration for DNN projects.

Phase of model	**%Freq**
Inference	**52.3**
Training	14.3
Benchmarking	7.1
Unknown	26.3

Results

Table 4.7 shows the result of the purpose of using hardware acceleration in the DNN projects. Based on our defined category for the proposition of using hardware acceleration in the DNN project, Inference, Training, and Benchmarking phases were found. Result demonstrates that 52.3% of the projects use hardware acceleration in the inference phase of the models, and 14.3% of the projects use hardware acceleration in the training phase of the models.
Model Training and **Model Inference** are the two primary steps of machine learning. During the training phase, a developer feeds their model a selected dataset to teach it what it needs to know about the sort of data it will evaluate. The model may then make predictions based on live data to provide actionable findings during the inference phase [111]. 7.1%

the projects use hardware acceleration in the benchmarking phase.

Model Benchmarking. The neural network's objective is to distinguish between Correct and Incorrect orbital mechanics. This is the standard by which we can compare the performance of various neural architectures. We can control how much disruption occurs because we are producing our data [112]. For 26.3% of the projects, there is no more information about the purposes of usage of hardware acceleration in the models and we put them in **Unknown** category.

Most of the projects leverage hardware acceleration for DNN inference while only a small portion of them leverage hardware acceleration for training DNN models. In addition, a small portion of projects use hardware acceleration for benchmarking purposes.

4.4 Discussions and Implications

Computer architectures are at the point of shifting from a general-purpose approach to a hardware/software co-design approach (i.e., domain specific architecture) [2]. To understand the practices and challenges of hardware-software co-design in the DNN acceleration context, in our study, we observe that DNN developers leverage hardware acceleration for a variety of DNN models (e.g., *CNN* and *RNN*). The studied projects mainly leverage hardware acceleration for *CNN* models, for the tasks such as *image classification* and *object detection*, which is indicate a high-level understanding of the status of hardware acceleration in the wild in real-world DNN projects.

We also observe that there are fewer developers working on HDLs than those working on GPPLs. Our observation indicates the imbalance between HDL developers and GPPLs developers. *Researchers* are the most popular category of developers who work on the DNN acceleration projects, while the developers contributing to GPPL code have more diverse profiles than those contributing to HDL code. It may explained by providing insights for improving such hardware-software co-design practices in the DNN hardware acceleration context and beyond. Finally, we observe that most of the projects leverage hardware acceleration for DNN *inference* while only a small portion of them leverage hardware acceleration for *training* DNN models. In addition, a small portion of projects use hardware acceleration for *benchmarking* purposes. **The software engineering community should leverage its experience in advancing GPPL development to help in advancing HDL development languages, providing HDL development tools and resources, and promoting community support in real world projects to have the balanced support for developing programs in GPPLs and HDLs.**

4.5 Threats to Validity

This work analyzes hardware acceleration for DNN projects on GitHub to understand how to use hardware acceleration for DNN projects. This work may not cover all projects related to DNN hardware acceleration. However, developers may share their DNN acceleration projects in other media. Other data sources could be used to expand our research in the future. In addition, we use a set of keywords to find and collect DNN acceleration projects from GitHub. Some DNN hardware acceleration keywords may have escaped our notice. In RQ1 and RQ3, we manually analyze model categories, tasks, and the purposes of using hardware acceleration for DNN in the related projects on GitHub. Our findings may be subjective and rely on the researchers' judgment during the manual analysis. To counteract the bias, two of the research authors collaborated on manual analysis and came to a significant agreement, showing the validity of the analytical results. To confirm the quality of the results, a third author participated in the discussions during the manual analysis.

4.6 Summary

In recent years, hardware acceleration for deep neural networks (DNNs) has been a popular domain-specific hardware-software co-design application. To further understand the techniques and challenges of HDL development in real-world projects, we look at 321 open-source projects linked to hardware acceleration for DNNs in our second study. We look at the types of projects that use DNN hardware acceleration, the programming languages and developer profiles used in these projects, and the goals of employing DNN hardware acceleration in these projects.

DNN developers leverage hardware acceleration for a variety of DNN models (e.g., CNN and RNN). The studied projects mainly leverage hardware acceleration for CNN models, for the tasks such as image classification and object detection. On the other hand, hardware acceleration is used much less frequently for RNNs. HDLs (e.g., VHDL) contribute to a higher portion of code than GPPLs (e.g., C) in the studied DNN acceleration projects. However, there are fewer developers working on HDLs than those working on GPPLs. Our observation indicates the imbalance between HDL developers and GPPLs developers. Researchers are the most popular category of developers who work on the DNN acceleration projects, while the developers contributing to GPPL code have more diverse profiles than those contributing to HDL code. Most of the projects leverage hardware acceleration for DNN inference while only a small portion of them leverage hardware acceleration for training DNN models. In addition, a small portion of projects use hardware acceleration for benchmarking purposes. Our results

can help practitioners and researchers understand the practices of DNN hardware acceleration in real-world projects, as well as the practices and challenges of HDL code development and software-hardware co-design in the DNN acceleration context.

CHAPTER 5 CONCLUSION

5.1 Summary

Studying the Challenges of Developing Hardware Description Language Programs consider the challenges that developers of HDLs face by examining HDL-related technical questions in the SO and EE forums. We observe that developing HDL programs face similar challenges as GPPL and challenges specific to HDLs (e.g., lower level operations such as bit and register operations or synchronization through clocked processes). We also observe that it is more challenging for developers of HDL programs to receive community support than GPPL (i.e., HDL-related questions are less likely and take longer to get accepted answers) which calls for efforts to enhance developer engagement or platform support in answering questions in technical forums (e.g., SO). In addition, our work identified opportunities for the research community to improve the practices of developing HDL programs, such as to improve the language abstraction for handling bit/register operations or file/memory I/Os, to provide actionable information to address syntax errors, and–or to develop tools to improve testing and simulation.

In recent years, hardware acceleration for deep neural networks (DNNs) has been a popular domain-specific hardware-software co-design application. To further understand the techniques and challenges of HDL development in real-world projects, we look at 321 open-source projects linked to hardware acceleration for DNNs in our second study. We look at the types of projects that use DNN hardware acceleration, the programming languages (GPPL and HDL programming languages) and developer profiles used in these projects (eight classifications identifier for developers), and the goals of employing DNN hardware acceleration in these projects(improve performance in inference and training phases). We observe that DNN developers leverage hardware acceleration for a variety of DNN models (e.g., CNN and RNN). The studied projects mainly leverage hardware acceleration for CNN models, for the tasks such as image classification and object detection. On the other hand, hardware acceleration is used much less frequently for RNNs. We also observe that HDLs (e.g., VHDL) contribute to a higher portion of code than GPPLs (e.g., C) in the studied DNN acceleration projects. However, there are fewer developers working on HDLs than those working on GPPLs. Our observation indicates the imbalance between HDL developers and GPPLs developers. In addition, we observe that most of the projects leverage hardware acceleration for DNN inference while only a small portion of them leverage hardware acceleration for training DNN models.

Our research discovered differences between developing HDL programs and general-purpose programming language (GPPL) programs and ways to improve HDL development. We hope that our study will call the attention of the software engineering community to the importance of HDL development and prevent HDLs from becoming a bottleneck in the development of future domain-specific computer systems.

5.2 Future Work

We hope to draw the SE community's attention to the importance of HDL program development which may become a bottleneck in developing domain-specific systems that are critical in increasing computing power in the future. We encourage researchers and practitioners in the SE community to leverage the advanced methodology, techniques, and tools in traditional software development to improve the practices of HDL program development. Future efforts are needed to help HDL developers address the most critical challenges (e.g., testing and simulation, syntax errors) in HDL development and the challenges specific to HDLs (e.g., process statements, bit operations). These topics represent the most challenging areas of HDL development and indicate opportunities for future work to improve the HDL development practices (e.g., to improve the language abstraction of handling file/memory I/Os, to provide actionable information to address syntax errors, or to develop tools to improve testing and simulation). Nevertheless, after searching available forums related to HDLs, we found that the most recent posts related to HDL questions on the internet are available on the studied SO and EE forums, while other forums with information related to HDL languages are not as active. Future work considering other data sources may complement our study. In addition, our identified and collected questions may not cover all the questions on Stack Overflow and Electrical Engineering forums.

Our other contribution is about the hardware accelerators for DNNs. We study the categories of projects that leverage hardware acceleration for DNN, the distribution of programming languages, developer profiles of these projects, and the purposes of using hardware acceleration for DNN in these projects. One future work can analyze the Hardware acceleration of other machine learning algorithms like SVM. In our study, projects in the GitHub repository are covered. Other repositories can be considered for similar analysis. In addition, the main challenges and issues of hardware-accelerated DNN can be identified to help developers and practitioners provide appropriate solutions. For this purpose, technical forums such as Stack Exchange can be used as a data source for identifying the main challenges and issue reports in this field.

REFERENCES

[1] S. Beyer, C. Macho, M. Di Penta, and M. Pinzger, "What kind of questions do developers ask on stack overflow? a comparison of automated approaches to classify posts into question categories," *Empirical Software Engineering*, vol. 25, no. 3, pp. 2258–2301, 2020.

[2] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Communications of the ACM*, vol. 62, no. 2, pp. 48–60, 2019.

[3] L. Truong and P. Hanrahan, "A golden age of hardware description languages: Applying programming language techniques to improve design productivity," in *3rd Summit on Advances in Programming Languages (SNAPL 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[4] J. Hennessy and D. Patterson, "A new golden age for computer architecture: Domain-specific hardware/software co-design, enhanced."

[5] I. C. society. Design automation technical committee, *IEEE Standard VHDL Language Reference Manual*. IEEE, 1988.

[6] I. S. Association *et al.*, "Ieee standard for verilog hardware description language (ieee 1364-2005)," *http://standards. ieee. org/*, 2006.

[7] J. I. Villar, J. Juan, M. J. Bellido, J. Viejo, D. Guerrero, and J. Decaluwe, "Python as a hardware description language: A case study," in *2011 VII Southern Conference on Programmable Logic (SPL)*. IEEE, 2011, pp. 117–122.

[8] M. Christensen, T. Sherwood, J. Balkind, and B. Hardekopf, "Wire sorts: a language abstraction for safe hardware composition," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2021, pp. 175–189.

[9] Hardware acceleration of deep neural network models on fpga. [Online]. Available: https://ignitarium.com/hardware-acceleration-of-deep-neural-network-models-on-fpga-part-1-of-2/

[10] M. Eleuldj *et al.*, "Survey of deep learning neural networks implementation on fpgas," in *2020 5th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech)*. IEEE, 2020, pp. 1–8.

[11] D. Harris and S. Harris, *Digital design and computer architecture*. Morgan Kaufmann, 2010.

[12] D. Thomas and P. Moorby, *The Verilog® hardware description language*. Springer Science & Business Media, 2008.

[13] P. J. Ashenden, *The designer's guide to VHDL*. Morgan Kaufmann, 2010.

[14] S. Sutherland, S. Davidmann, and P. Flake, *SystemVerilog for Design Second Edition: A Guide to Using SystemVerilog for Hardware Design and Modeling*. Springer Science & Business Media, 2006.

[15] S. Bailey, "Comparison of VHDL, Verilog and SystemVerilog," p. 29, 2003.

[16] tensorflow. [Online]. Available: https://www.tensorflow.org/model_optimization

[17] Deploy ml models to field-programmable gate arrays (fpgas) with azure machine learning. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-fpga-web-service

[18] L. W. Wills and K. Swaminathan, "Guest editorial: Ieee tc special issue on domain-specific architectures for emerging applications," *IEEE Transactions on Computers*, vol. 69, no. 08, pp. 1096–1098, 2020.

[19] W. J. Dally, Y. Turakhia, and S. Han, "Domain-specific hardware accelerators," *Communications of the ACM*, vol. 63, no. 7, pp. 48–57, 2020.

[20] N. P. Jouppi, C. Young, N. Patil, and D. Patterson, "A domain-specific architecture for deep neural networks," *Communications of the ACM*, vol. 61, no. 9, pp. 50–59, 2018.

[21] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[22] A. Ankit, I. El Hajj, S. R. Chalamalasetti, S. Agarwal, M. Marinella, M. Foltin, J. P. Strachan, D. Milojicic, W.-M. Hwu, and K. Roy, "Panther: A programmable architecture for neural network training harnessing energy-efficient reram," *IEEE Transactions on Computers*, vol. 69, no. 8, pp. 1128–1142, 2020.

[23] T. Wang, T. Geng, A. Li, X. Jin, and M. Herbordt, "Fpdeep: Scalable acceleration of cnn training on deeply-pipelined fpga clusters," *IEEE Transactions on Computers*, vol. 69, no. 8, pp. 1143–1158, 2020.

[24] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead," *IEEE Access*, vol. 8, pp. 225 134–225 180, 2020.

[25] F. Turan, S. S. Roy, and I. Verbauwhede, "Heaws: an accelerator for homomorphic encryption on the amazon aws fpga," *IEEE Transactions on Computers*, vol. 69, no. 8, pp. 1185–1196, 2020.

[26] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. A. Horowitz, "Convolution engine: balancing efficiency & flexibility in specialized computing," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, 2013, pp. 24–35.

[27] B. L. West, J. Zhou, R. G. Dreslinksi, O. D. Kripfgans, J. B. Fowlkes, C. Chakrabarti, and T. F. Wenisch, "Tetris: Using software/hardware co-design to enable handheld, physics-limited 3d plane-wave ultrasound imaging," *IEEE Transactions on Computers*, vol. 69, no. 8, pp. 1209–1220, 2020.

[28] S. E. Arda, A. Krishnakumar, A. A. Goksoy, N. Kumbhare, J. Mack, A. L. Sartor, A. Akoglu, R. Marculescu, and U. Y. Ogras, "Ds3: A system-level domain-specific system-on-chip simulation framework," *IEEE Transactions on Computers*, vol. 69, no. 8, pp. 1248–1262, 2020.

[29] Y. Turakhia, G. Bejerano, and W. J. Dally, "Darwin: A genomics co-processor provides up to 15,000 x acceleration on long read assembly," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 199–213, 2018.

[30] A. Addisie and V. Bertacco, "Collaborative accelerators for streamlining mapreduce on scale-up machines with incremental data aggregation," *IEEE Transactions on Computers*, vol. 69, no. 8, pp. 1233–1247, 2020.

[31] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a Scala embedded language," in *DAC Design Automation Conference 2012*. IEEE, 2012, pp. 1212–1221.

[32] "Magma," https://github.com/phanrahan/magma, Last accessed 24/10/2021.

[33] P. Bjesse, K. Claessen, M. Sheeran, and S. Singh, "Lava: hardware design in haskell," *ACM SIGPLAN Notices*, vol. 34, no. 1, pp. 174–184, 1998.

[34] Y. Li and M. Leeser, "Hml, a novel hardware description language and its translation to vhdl," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 1, pp. 1–8, 2000.

[35] J. Decaluwe, "Myhdl: a python-based hardware description language." *Linux journal*, no. 127, pp. 84–87, 2004.

[36] D. Galloway, "The transmogrifier c hardware description language and compiler for fpgas," in *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines*. IEEE, 1995, pp. 136–144.

[37] Y. Endoh, "Asystemc: an aop extension for hardware description language," in *Proceedings of the tenth international conference on Aspect-oriented software development companion*, 2011, pp. 19–28.

[38] J. O'Leary, M. Linderman, M. Leeser, and M. Aagaard, "Hml: A hardware description language based on standard ml," in *Computer Hardware Description Languages and their applications*. Elsevier, 1993, pp. 327–334.

[39] D. C. Black, J. Donovan, B. Bunton, and A. Keist, *SystemC: From the ground up*. Springer Science & Business Media, 2009, vol. 71.

[40] R. Daly, L. Truong, and P. Hanrahan, "Invoking and linking generators from multiple hardware languages using coreir," in *Second Workshop on Open-Source EDA Technology (WOSET)*, 2018.

[41] A. Izraelevitz, J. Koenig, P. Li, R. Lin, A. Wang, A. Magyar, D. Kim, C. Schmidt, C. Markley, J. Lawson *et al.*, "Reusability is firrtl ground: Hardware construction languages, compiler frameworks, and transformations," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 209–216.

[42] C. Wolf, "Yosys open synthesis suite," 2016.

[43] T. Heldmann, T. Schneider, O. Tkachenko, C. Weinert, and H. Yalame, "Llvm-based circuit compilation for practical secure computation," in *International Conference on Applied Cryptography and Network Security*. Springer, 2021, pp. 99–121.

[44] S.-H. Wang, A. Sridhar, and J. Renau, "Lnast: A language neutral intermediate representation for hardware description languages," in *Proc. 2nd Workshop Open-Source EDA Technol.*, 2019.

[45] "Circuit ir compilers and tools," https://circt.llvm.org, Last accessed 24/10/2021.

[46] W. Snyder, "Verilator and systemperl," in *North American SystemC Users' Group, Design Automation Conference*, 2004.

[47] C. Elliott, "Compiling to categories," in *Proceedings of the ACM on Programming Languages (ICFP)*, 2017. [Online]. Available: http://conal.net/papers/compiling-to-categories

[48] "Qusoc - low-level rtl design using c and quokka fpga toolkit," https://github.com/EvgenyMuryshkin/qusoc, Last accessed 24/10/2021.

[49] "Academic papers using stack exchange data," https://meta.stackexchange.com/questions/134495/academic-papers\-using-stack-exchange-data, Last accessed 24/10/2021.

[50] M. Allamanis and C. Sutton, "Why, when, and what: analyzing stack overflow questions by topic, type, and code," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 53–56.

[51] C. Rosen and E. Shihab, "What are mobile developers asking about? a large scale study using stack overflow," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1192–1223, 2016.

[52] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web?(nier track)," in *Proceedings of the 33rd international conference on software engineering*, 2011, pp. 804–807.

[53] S. Beyer and M. Pinzger, "A manual categorization of android app development issues on stack overflow," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 531–535.

[54] S. Beyer, C. Macho, M. Di Penta, and M. Pinzger, "Analyzing the relationships between android api classes and their references on stack overflow," *Technical Report*, 2017.

[55] L. S. Barbosa, "Software engineering for'quantum advantage'," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 427–429.

[56] S. Wang, D. Lo, and L. Jiang, "An empirical study on developer interactions in stackoverflow," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 2013, pp. 1019–1024.

[57] M. Linares-Vásquez, B. Dit, and D. Poshyvanyk, "An exploratory analysis of mobile development issues using stack overflow," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 93–96.

[58] P. K. Venkatesh, S. Wang, F. Zhang, Y. Zou, and A. E. Hassan, "What do client developers concern when using web apis? an empirical study on developer forums and stack overflow," in *2016 IEEE International Conference on Web Services (ICWS)*. IEEE, 2016, pp. 131–138.

[59] S. Ahmed and M. Bagherzadeh, "What do concurrency developers ask about? a large-scale study using stack overflow," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018, pp. 1–10.

[60] X.-L. Yang, D. Lo, X. Xia, Z.-Y. Wan, and J.-L. Sun, "What security questions do developers ask? a large-scale study of stack overflow posts," *Journal of Computer Science and Technology*, vol. 31, no. 5, pp. 910–924, 2016.

[61] M. Bagherzadeh and R. Khatchadourian, "Going big: a large-scale study on what big data developers ask," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 432–442.

[62] M. Rebouças, G. Pinto, F. Ebert, W. Torres, A. Serebrenik, and F. Castor, "An empirical study on the usage of the Swift programming language," in *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 634–638.

[63] G. Uddin, F. Sabir, Y.-G. Guéhéneuc, O. Alam, and F. Khomh, "An empirical study of iot topics in iot developer discussions on stack overflow," *Empirical Software Engineering*, vol. 26, no. 6, pp. 1–45, 2021.

[64] J. Han, E. Shihab, Z. Wan, S. Deng, and X. Xia, "What do programmers discuss about deep learning frameworks," *Empirical Software Engineering*, vol. 25, pp. 2694–2747, 2020.

[65] J. Zou, L. Xu, M. Yang, X. Zhang, and D. Yang, "Towards comprehending the non-functional requirements through developers' eyes: An exploration of stack overflow using topic analysis," *Information and Software Technology*, vol. 84, pp. 19–32, 2017.

[66] J. Zou, L. Xu, W. Guo, M. Yan, D. Yang, and X. Zhang, "Which non-functional requirements do developers focus on? an empirical study on stack overflow using topic analysis," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 446–449.

[67] Y. Zhang, D. Lo, X. Xia, and J.-L. Sun, "Multi-factor duplicate question detection in stack overflow," *Journal of Computer Science and Technology*, vol. 30, no. 5, pp. 981–997, 2015.

[68] C. Treude and M. Wagner, "Predicting good configurations for github and stack overflow topic models," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 84–95.

[69] T. Moreau, Z. J. Tianqi Chen, and A. K. Luis Ceze, Carlos Guestrin, "Vta: An open hardware-software stack for deep learning," *ArXiv*, vol. abs/1807.04188, 2018.

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

[71] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[72] F. Vaverka, V. Mrazek, Z. Vasicek, and L. Sekanina, "Tfapprox: Towards a fast emulation of dnn approximate hardware accelerators on gpu," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 294–297.

[73] V. Mrazek, L. Sekanina, and Z. Vasicek, "Using libraries of approximate circuits in design of hardware accelerators of deep neural networks," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2020, pp. 243–247.

[74] C. Mackin, H. Tsai, S. Ambrogio, P. Narayanan, A. Chen, and G. W. Burr, "Weight programming in dnn analog hardware accelerators in the presence of nvm variability," *Advanced Electronic Materials*, vol. 5, no. 9, p. 1900026, 2019.

[75] J. Loh, J. Wen, and T. Gemmeke, "Low-cost dnn hardware accelerator for wearable, high-quality cardiac arrythmia detection," in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2020, pp. 213–216.

[76] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2019, pp. 304–315.

[77] G. Li, S. K. 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 *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[78] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of dnn accelerators using scale-sim," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2020, pp. 58–68.

[79] M. I. M. Collantes, Z. Ghodsi, and S. Garg, "Safetpu: A verifiably secure hardware accelerator for deep neural networks," in *2020 IEEE 38th VLSI Test Symposium (VTS)*. IEEE, 2020, pp. 1–6.

[80] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[81] H. Zeng, R. Chen, C. Zhang, and V. Prasanna, "A framework for generating high throughput cnn implementations on fpgas," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2018, pp. 117–126.

[82] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to fpgas," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[83] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "An automatic rtl compiler for high-throughput fpga implementation of diverse deep convolutional neural networks," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.

[84] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2017, pp. 152–159.

[85] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnbuilder: An automated tool for building high-performance dnn hardware accelerators for fpgas," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[86] D. H. Noronha, B. Salehpour, and S. J. Wilton, "Leflow: Enabling flexible fpga high-level synthesis of tensorflow deep neural networks," in *FSP Workshop 2018; Fifth International Workshop on FPGAs for Software Programmers*. VDE, 2018, pp. 1–8.

[87] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[88] A. Shawahna, S. M. Sait, and A. El-Maleh, "Fpga-based accelerators of deep learning networks for learning and classification: A review," *ieee Access*, vol. 7, pp. 7823–7859, 2018.

[89] Open source fpga accelerator projects. [Online]. Available: https://github.com/search?q=fpga+accelerator

[90] S. Baltes, L. Dumani, C. Treude, and S. Diehl, "Sotorrent: Reconstructing and analyzing the evolution of stack overflow posts," in *Proceedings of the 15th international conference on mining software repositories*, 2018, pp. 319–330.

[91] "Stack Exchange Data Explorer," https://data.stackexchange.com, Last accessed 04/04/2022.

[92] M. Alshangiti, H. Sapkota, P. K. Murukannaiah, X. Liu, and Q. Yu, "Why is developing machine learning applications challenging? a study on stack overflow posts," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2019, pp. 1–11.

[93] M. Openja, B. Adams, and F. Khomh, "Analysis of modern release engineering topics:– a large-scale study using stackoverflow–," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2020, pp. 104–114.

[94] K. S. Jones and P. Willett, *Readings in information retrieval*. Morgan Kaufmann, 1997.

[95] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[96] A. K. McCallum, "Mallet: A machine learning for language toolkit," *http://mallet. cs. umass. edu*, 2002.

[97] C.-M. Tan, Y.-F. Wang, and C.-D. Lee, "The use of bigrams to enhance text categorization," *Information processing & management*, vol. 38, no. 4, pp. 529–546, 2002.

[98] M. Röder, A. Both, and A. Hinneburg, "Exploring the space of topic coherence measures," *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, 2015.

[99] T. Zimmermann, "Card-sorting: From text to themes," in *Perspectives on data science for software engineering*. Elsevier, 2016, pp. 137–141.

[100] H. Li, W. Shang, B. Adams, M. Sayagh, and A. E. Hassan, "A qualitative study of the benefits and costs of logging from developers' perspectives," *IEEE Transactions on Software Engineering*, 2020.

[101] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[102] M. Openja, B. Adams, and F. Khomh, "Analysis of modern release engineering topics : – a large-scale study using stackoverflow –," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020, pp. 104–114.

[103] R. Zhao, W. Luk, X. Niu, H. Shi, and H. Wang, "Hardware acceleration for machine learning," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 645–650.

[104] Pipecnn. [Online]. Available: https://github.com/doonny/PipeCNN

[105] spoonn. [Online]. Available: https://github.com/fpgasystems/spooNN

[106] brain. [Online]. Available: https://github.com/ReconfigureIO/brain

[107] Gyro. [Online]. Available: https://github.com/federicohyo/Gyro

[108] Image classification using cnn on fpga. [Online]. Available: https://github.com/padhi499/Image-Classification-using-CNN-on-FPGA

[109] Infernce. [Online]. Available: https://github.com/bubblebeam/Inferno-Realtime-Fire-detection-using-CNNs

[110] Ece 5775. [Online]. Available: https://github.com/shivarajagopal/ece5775-final

[111] deep-learning-training-and-inferenc. [Online]. Available: https://www.xilinx.com/applications/ai-inference/difference-between-deep-learning-training-and-inference.html

[112] benchmarks. [Online]. Available: https://towardsdatascience.com/neural-network-benchmarks-82d48425c21b