Advancing HCI with Neuromorphic Technology: Guidelines for Designing User-Friendly Developer Tools for Neuromorphic Development

Divesh Upreti
George Mason University
Fairfax, Virginia, USA
dupreti@gmu.edu

Aditi Maheshwari
Accenture Labs
San Francisco, California, USA
aditi.maheshwari@accenture.com

Taylor Tabb
Accenture Labs
San Francisco, California, USA
taylor.tabb@accenture.com

Ioannis Polykretis
Accenture Labs
Accenture
San Francisco, California, USA
ioannis.polykretis@accenture.com

Eric M Gallo
Accenture Labs
Accenture
San Francisco, California, USA
eric.gallo@accenture.com

Kenneth Michael Stewart
Future Technologies Group
Accenture Labs
San Francisco, California, USA
kennetms@uci.edu

Thomas D. LaToza
Department of Computer Science
George Mason University
Fairfax, Virginia, USA
tlatoza@gmu.edu

Andreea Danielescu
Accenture Labs
San Francisco, California, USA
andreea.danielescu@accenture.com

Abstract

Neuromorphic technology offers advantages such as low-power processing, low latency, adaptive learning, and noise tolerance, making it ideal for edge computing applications. However, developers face significant hurdles due to the nascent nature of the field, including limited access to hardware and software, lack of benchmarks, and the need for deep interdisciplinary knowledge. Through interviews with 12 practitioners from both industry and academia, we conducted a thematic analysis to understand the current landscape of neuromorphic programming and identified key challenges, workflows, and potential solutions for enhancing accessibility and adoption. Our findings led to a set of guidelines for creating more accessible software development tools and platforms for those looking to create neuromorphic applications. Through this work, we aim to bridge the gap between neuromorphic computing and the HCI community, promoting the design of more intuitive and effective interfaces for neuromorphic development, and ultimately facilitating the creation of edge intelligent systems.

CCS Concepts

• **Computing Methodologies** → **Bio Inspired Approaches**; • **Hardware** → **Neural systems**.

Keywords

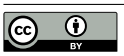
neuromorphic computing, developer experience, developer tools, neural networks

ACM Reference Format:
Divesh Upreti, Aditi Maheshwari, Taylor Tabb, Ioannis Polykretis, Eric M Gallo, Kenneth Michael Stewart, Thomas D. LaToza, and Andreea Danielescu. 2025. Advancing HCI with Neuromorphic Technology: Guidelines for Designing User-Friendly Developer Tools for Neuromorphic Development. In *CHI Conference on Human Factors in Computing Systems (CHI '25), April 26–May 01, 2025, Yokohama, Japan.* ACM, New York, NY, USA, Article 111, 18 pages. https://doi.org/10.1145/3706598.3713249

1 Introduction

With intelligent systems becoming increasingly sophisticated and highly specialized, heterogeneous computing, especially at the edge, is becoming the norm [18, 80]. An emerging component of the heterogeneous computing landscape is neuromorphic computing (NC) [2, 48, 59]. Neuromorphic technology includes hardware and software systems that replicate the structure and functionality of biological neural networks and bring distinct advantages and capabilities like extremely low-power neural processing, low latency, adaptive learning, and noise tolerance. As we progress towards a world where computational materials and distributed IoT sensor networks will penetrate every aspect of human life [4], neuromorphic technology is poised to play a crucial role in overcoming key

Authors' Contact Information: Divesh Upreti, George Mason University, Fairfax, Virginia, USA, dupreti@gmu.edu; Aditi Maheshwari, Accenture Labs, San Francisco, California, USA, aditi.maheshwari@accenture.com; Taylor Tabb, Accenture Labs, San Francisco, California, USA, taylor.tabb@accenture.com; Ioannis Polykretis, Accenture Labs
Accenture, San Francisco, California, USA, ioannis.polykretis@accenture.com; Eric M Gallo, Accenture Labs
Accenture, San Francisco, California, USA, eric.gallo@accenture.com; Kenneth Michael Stewart, Future Technologies Group
Accenture Labs, San Francisco, California, USA, kennetms@uci.edu; Thomas D. LaToza, Department of Computer Science
George Mason University, Fairfax, Virginia, USA, tlatoza@gmu.edu; Andreea Danielescu, Accenture Labs, San Francisco, California, USA, andreea.danielescu@accenture.com.



CHI '25, Yokohama, Japan

ACM ISBN 979-8-4007-1394-1/25/04
https://doi.org/10.1145/3706598.3713249

constraints around power, latency, sustainability, data security, and computational complexity. With the potential to create systems that collect, process, and analyze complex data locally in real time with high power efficiency, and the versatility to deploy neuromorphic processing in both silicon and non-silicon materials using digital or analog approaches, neuromorphic technology will be key to realizing Weiser's vision of ubiquitous computing [70]. NC differs fundamentally from conventional Von Neumann architectures in a number of ways, as shown in Figure 1. Von Neumann architectures use binary encoding (1's and 0's) to process information, and computation is done serially in discrete, clock-controlled time steps. Processing and memory are physically separate, requiring constant data transfer between them. Conversely, neuromorphic technologies – like neuromorphic processors [66], mimic the architecture of biological processing by encoding information through the number or timing of spikes and by processing the information through asynchronous networks of neurons and synapses. Memory and processing are co-located, eliminating transfer bottlenecks. This distinctive architecture enables high parallelism and event-driven processing – where only a small portion of the system is active at any given time – resulting in low latency and power consumption, which are both ideal for edge devices.

Neuromorphic technology's inherent neural network-style computation makes it a natural platform for many of today's artificial intelligence (AI) tasks [58]. Sharing overlapping applications and methods with machine learning (ML) and deep learning (DL), [55, 56], NC attracts developers traditionally focused on artificial neural networks (ANN). Originating from diverse backgrounds, developers enter the NC field facing a yet emerging, but already complex and 'full-stack' technological domain that requires deep knowledge of neuroscience, computer science, and hardware design to tackle its unique hardware and software challenges. This knowledge is crucial to also allow developers to leverage existing commercial neuromorphic technology such as event-driven cameras [1] for highly specialized applications. However, upon entering the field, developers face a variety of domain-related complexities such as limited access to existing hardware and software, scarcity of benchmarks and metrics, minimal abstraction, and having to catch up to performance benchmarks that have been rigorously set and continuously one-upped by traditional AI tools, which prioritize accuracy over other key metrics such as power consumption or adaptability [58]. Consequently, learning how to develop and deploy applications may seem insurmountable for those not already deeply embedded in the field. Owing to the significant overlap of NC with ML, efforts have been made to overcome these challenges, by drawing inspiration from successful developer tools used in traditional AI and ML applications [28, 29], or outlining opportunities for the neuromorphic community to foster algorithmic and application development [58]. While being only the early steps in opening neuromorphic development to the masses, these efforts highlight the importance of seeking inspiration and lessons from processes, tools, and experiences in traditional neural network development to form a more nuanced perspective on how to solve existing challenges in neuromorphic development and identify where the technology can be most suitably applied.

Aiming to further structure these efforts and bridge the gap between NC and HCI, we interviewed 12 expert practitioners with varying years of experience in neuromorphic development to understand the motivations, workflows, challenges, and potential solutions towards improving the accessibility and broader adoption of this technology. The goal of our study is to explore how HCI researchers can contribute to making NC more usable and approachable, and embed it within HCI contexts. HCI researchers, with their expertise in usability, developer experience, and tool design, are uniquely positioned to address the challenges around the adoption of NC. Therefore, HCI researchers and practitioners focused on designing developer tools are the key audience for this work.

Well designed developer tools are crucial for translating NC's potential into practical applications within HCI, especially for emerging technologies such as intelligent wearables, e-textiles, IoT systems, distributed sensor networks, autonomous robots, and many more. These domains demand computing solutions that prioritize low-power, low-latency, and personalized functionality, which NC can uniquely provide. Current ubiquitous computing systems in HCI, such as IoT devices and distributed sensor networks, often rely on conventional computing architectures that are inherently power-intensive, limiting their scalability and efficiency in real-time, energy-constrained environments. This high energy consumption poses significant challenges to the deployment of sustainable, low-power ubiquitous computing solutions [16]. By addressing the limitations of current systems, NC has the potential to redefine the capabilities of ubiquitous computing, making it a critical area of exploration for HCI researchers aiming to develop sustainable and efficient computing solutions [10]. More broadly, NC aligns with three core themes in HCI: advancing ubiquitous computing through energy-efficient, scalable systems; driving sustainability with low-power solutions that integrate with energy-harvesting technologies; and enabling novel applications, such as adaptive and hyper-personalized wearables, distributed environmental sensors, and intelligent systems.

Through a thematic analysis of our interview data, we highlight experts' sentiments around NC, commonly utilized workflows, and existing challenges and gaps in neuromorphic development. We leverage these insights to develop a set of guidelines for creating user-friendly tools and platforms for neuromorphic development and highlight important advancements in neuromorphic tool development and learnings that could be borrowed from traditional developer tools to inform future directions for neuromorphic development. By addressing the challenges and identifying opportunities for HCI researchers focused on developer tooling, we provide actionable guidelines to lower barriers to NC adoption, making it a valuable resource for HCI researchers invested in the future of computing.

In summary, our primary contributions are:

- An in-depth exploration of the motivations, workflows, tools and resources currently used, and pain points associated with neuromorphic application development, along with helpful prerequisites for getting started with NC, and key insights around areas of improvement in neuromorphic development gathered through interviews with 12 experts in the field (Section 4).

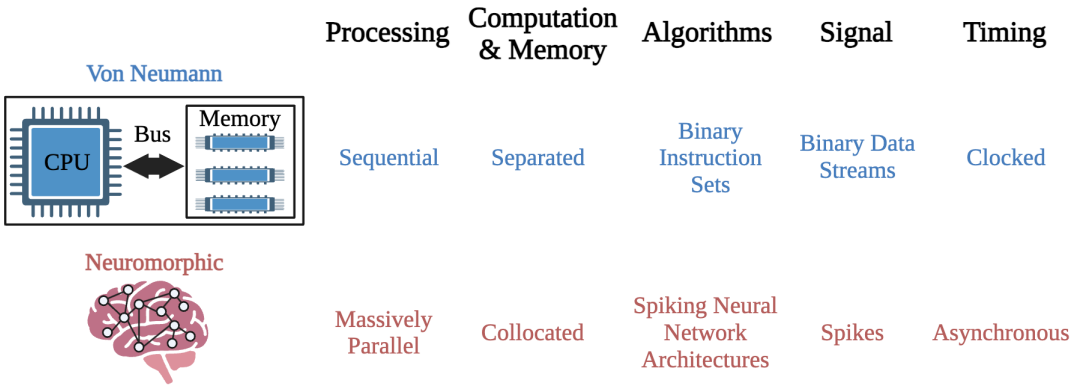


Figure 1: Comparison of the computational principles governing Von Neumann and neuromorphic architectures

- Opportunities and actionable guidelines for creating user-friendly and intuitive tools and platforms for neuromorphic development, specifically for HCI researchers focused on developer tools. These recommendations leverage insights and best practices from adjacent computing domains, such as machine learning and traditional software development, to address the unique challenges of neuromorphic computing (Section 5).

2 Background

2.1 Developer Tools for Neuromorphic and Traditional Machine Learning Systems

Current tooling and support software for neuromorphic systems can be categorized into five main types [60]:

- **Custom hardware synthesis tools**: Such tools convert a higher-level network description into a low-level neural circuitry representation (e.g., SpiNNaker [51]).
- **Mapping and Programming Tools**: Mapping tools map an existing neural network model onto a neuromorphic architecture, while programming tools allow users to directly program particular neuromorphic architectures (e.g., Lava [23]).
- **Simulators**: Neuromorphic simulators are essential development tools to emulate neuromorphic hardware performance in a non-neuromorphic system [62]. Simulations are performed at various scales ranging from subcellular, to single neurons, and from small custom networks to multilayer architectures. Simulations also examine the performance at a range of levels: from the software to the single chip and the whole system. They are used for verifying the performance of neuromorphic hardware, developing and training algorithms [60], and building a user base for not-readily-available hardware. Simulation tools like Nest [31], Brian [32], and Nengo [11] have been previously studied for general usability and effectiveness.
- **Libraries and Frameworks**: A wide range of software systems have been developed to streamline and standardize the developpent of neuromorphic applications through reusable and optimized code bases. These range from platform-agnostic tools such as SNNtorch [28] and SpikingJelly [29] that adapt the structure of well-established ML frameworks like Pytorch to neuromoprhic needs, to platform-specific tools such as Lava [23] and Rockpool [49] that aim for deployment on specific hardware.
- **Visualization tools**: These tools offer a more intuitive insight of the inner workings of neuromorphic systems[27] [38] (e.g., Nest [31], BindsNET [33], Brian [32], Brian2GeNN [64], Brian 2 [63] and Nengo [11]).

However, there is a significant challenge with this arsenal of tools: although some systems like Nengo [11] and SNNtorch [28] aim to cross the boundaries of specific platforms, they are usually not end-to-end from software design to hardware deployment. When they are, they tend to be hardware-specific and limited to the manufacturers that developed them [15, 49], making them less accessible and applicable outside specific use cases and communities [58].

The development environments, tools, frameworks, and libraries used in neuromorphic programming draw some inspiration from those used in conventional ML, and especially deep learning (DL). Due to the underlying neural network architectures of both DL and NC, these two approaches share many methodological similarities in terms of neuron modeling [56], debugging, parameter optimization, and resource availability [20, 79]. Hyperparameter selection and optimization is a primary challenge in developing both neuromorphic systems [20] and traditional ML [82], since parameters such as learning rates, regularization coefficients, and initialization strategies significantly influence the learning dynamics and system performance. Additionally, NC and DL both involve building and training neural networks using various datasets, whose quality heavily affects the effectiveness and correctness of the models [81].

Several studies have investigated these challenges and their impact on developer workflows. Developers lack key knowledge, preventing them from making progress, including practical knowledge of the APIs through which developers interact with frameworks and libraries [19]. More fundamentally, developers lack knowledge of the conceptual underpinnings of the technology, including their key ideas, and the ability to use these concepts to build the right conceptual models to effectively write and debug code. ANN systems also require new skills in discovering, managing, and versioning

Aspect of Comparison	ANN Programming	Neuromorphic Programming
Frameworks and Libraries	TensorFlow, PyTorch, Keras, scikit-learn	NEST [31], Brian2 [32], GeNN [75]
Deployment Platforms	Deployable on various platforms (cloud, edge, mobile) [68]	Specialized neuromorphic hardware platforms [25]
Hardware Integration	Hardware-agnostic, CPU, GPU deployment	Integration with Neuromorphic Hardware, Event-driven processors [25]
Real-time Processing	Primarily designed for batch processing	Emphasis on real-time processing and low-latency responses [58]
Community Support	Large and active community support for mainstream frameworks	Smaller, more specialized community, specific to neuromorphic programming [60]

Table 1: Comparison of key aspects between artificial neural network (ANN) programming and neuromorphic programming, including frameworks, deployment platforms, hardware integration, processing capabilities, and community support.

data; require different techniques for customizing and reusing models; and are harder to work with as models behave in erratic and unpredictable ways [6]. Most developer questions focus on data preparation and model setup necessary to correctly create a ML pipeline [30], with particular challenges around API misuse, hyperparameter selection, GPU computation, and limited support for debugging and profiling [81]. As a result of this complexity and missing knowledge, many non expert developers simply reuse existing models, with less knowledge of their inner workings [74].

Despite the similarities with ANNs, NC faces its own set of unique challenges due to the fundamental difference between neuromorphic and ML systems in terms of spikes vs. bits, which is a new concept for most developers [5]. This further amplifies the challenge of a lack of conceptual understanding, as even the most basic concepts about how programs execute suddenly requires knowledge developers lack. Moreover, traditional computing possesses a comprehensive set of development tools, and a large number of users trained in utilizing such tools [24]. In contrast, NC has the added challenge of having to vie for attention and resources from the researchers and users of conventional architectures [58].Without the scale and maturity of ANN frameworks and libraries, NC developers lack sufficient pre-built models to fall back on, requiring developers to instead carefully craft solutions to routine problems for which ANN developers would already have examples. A summary of the similarities and differences between ANN and Neuromorphic Programming is provided in Table 1.

The advantages of NC, such as energy efficiency, low latency and robustness, constitute sufficient incentives for developers to work with a different and unconventional architecture [12], but dedicated efforts are needed to understand how neuromorphic development can be made more broadly accessible to software engineers and how barriers to entry can be reduced. More established specialized programming domains have benefited greatly from specialized tool support, substantially simplifying programming tasks. For example, the live programming and reproducibility of Computational Notebooks make data analysis far easier [39]. Building software with transient contributors through self-contained microtasks is made possible through programming tools which carefully minimize work inter-dependencies [41]. By shedding light on the key sources of complexity in understanding and working with NC, we aim to inspire future HCI researchers to similarly simplify NC programming through new programming interactions.

2.2 Studies on Challenges around Developing Neuromorphic Systems

Past research shows that the lack of readily accessible and usable software, simulators, and hardware systems is a key issue that inhibits software development in the neuromorphic domain [58]. The lack of resources results in longer training times when compared to non-spiking techniques, highly hardware-specific software, and simulators that do not effectively scale and which are unable to keep up with the use of large data sets. The authors propose that increasing the accessibility of hardware and improving the effectiveness of simulators will help developers more quickly evaluate their algorithms, resulting in faster neuromorphic algorithm development. Further, a previous survey around NC [60] shows that, in comparison to the volume of hardware implementations of neuromorphic systems, there has been little work focused on the development of supporting software. The study asserts that more work must be done on supporting software alongside hardware improvements to benefit the neuromorphic community. Our study explores the challenges faced by neuromorphic developers, analyzing expert insights to propose practical solutions, design guidelines for development tools, and improvements in developer support, drawing parallels to advancements in deep learning and aiming to create a clearer learning path for newcomers to NC.

3 Methods

3.1 Participants

We recruited 12 participants from the niche scientific neuromorphic community in the spring of 2022 through targeted recruitment efforts and snowball sampling. We primarily focused on researchers with a track record of strong technical work or publications in the space, as well as relevant industry experience. We also included junior researchers with over a year of experience in neuromorphic algorithm or application development. 5 of our participants worked in research labs or the IT industry, while the remainder worked in a university setting. 1 participant was an undergraduate student, 3 were PhD students, 2 were post-doctoral researchers, and 1 was faculty. Participants varied in programming experience from 5 to 39 years, with a median of 10 years. They had a median of 7 years of experience with neuromorphic development, ranging from 1 to 11 years. They came from diverse academic and professional backgrounds, including electrical and computer engineering, civil

engineering, computational neuroscience, and cognitive science, and were based in regions across North America, Europe, and Australia. See Table 2 for a summary of the participant demographics.

While our participants varied in their years of experience, they were all recognized experts in the field. The participants in our study have either developed or supported existing NC tools, or were collaborators and students of prominent members of the NC community. Collectively, they represent a group with firsthand knowledge of the current state of NC development, tooling, and challenges.

3.2 Interviews

To guide our exploration of the challenges and opportunities in neuromorphic development, we framed our interview questions around the following research questions:

(1) What motivates developers to adopt neuromorphic computing, and how do these motivations shape their workflows and priorities?
(2) What are the key workflows, tools, and resources currently used for neuromorphic development, and what challenges do developers encounter when employing them?
(3) What are the primary challenges and pain points faced by neuromorphic developers, and how can these inform the design of more effective tools and systems?
(4) What skills, resources, and strategies are essential for newcomers to neuromorphic computing, and how can these be better supported through tool and system design?

We began by obtaining informed consent from participants before conducting the semi-structured interviews. After gathering information about their current role, experience, and preferences for development platforms and languages, participants were asked about how and why they built any recent neuromorphic algorithms or applications. Next, they were asked to talk through and explain their development workflow. Participants were then asked to identify the challenges they have faced with neuromorphic programming, particularly those that differed from traditional programming. Participants were prompted to reflect on challenges related to development time, optimization, resource availability, adapting to neuromorphic architectures, and any other challenges they had experienced. Finally, participants were asked to recommend any solutions to these challenges, and how learning can be made easier for beginners. For a full list of planned interview questions, see Appendix A. Participants were not compensated for their time, as participation was voluntary.

Interviews were conducted in April of 2022 through Zoom and the participants were interviewed for approximately 37 to 90 minutes, with a median interview length of 53 minutes. We transcribed and anonymized interviews using Otter.ai [52] and manually corrected any errors in the transcription before proceeding with qualitative coding. Our goal was to keep the interview questions fairly broad to allow participants to share their unique experiences in neuromorphic computing and allow deeper exploration into specific challenges that interested the participants. Due to the nature of semi-structured interviews, tangential responses were entertained and discussions were eventually guided back with follow-up questions to focus on the challenges in and recommendations for NC Development. This flexible approach contributed to the variation in the length of study sessions.

NC is a highly specialized field within neuroscience and computer science and engineering, with the majority of research activity concentrated in major universities and research labs globally. Experts estimate the number of active researchers in NC worldwide to be in the low thousands. As such, the pool of individuals with deep expertise in this area is small, and shared experiences and challenges are common among practitioners. Consequently, our interviews did not reveal significant variability in perspectives, which we attribute to the niche nature of the field rather than the study design.

This shared understanding among participants strengthens the reliability of the findings, as it highlights the consistency of the challenges and opportunities faced by the community. The participants' collective expertise provides valuable insights into the motivations, workflows, and challenges associated with NC development, enabling us to identify specific areas for improvement in NC tooling and practices.

3.3 Analysis

After anonymizing the transcripts, we imported them into Atlas.ti [8], a tool for qualitative coding. The data from pilot interviews was used to generate initial inductive codes, such as workflow bottlenecks, challenges in simulation, tooling gaps, and common challenges in transitioning to neuromorphic systems, which helped refine the semi-structured interview design. Three of the authors then developed a coding schema containing deductive codes generated from the original set of planned interview questions. The coding schema was reviewed and finalized for coding the remaining interviews through consensus among the coders. Any disagreements in code assignment, grouping, or phrasing were resolved through discussion. During the coding process, the schema was iteratively refined—codes were condensed, clarified, clustered, and rephrased. New codes were identified inductively through an interpretive analysis of the transcript data. Whenever a new code was agreed upon, the coders revisited prior transcripts to determine its presence, achieved consensus, and updated the coding as necessary. This iterative process resulted in a total of 39 codes and 7 code groups, which were further organized into 5 overarching themes, as described in the next section. For more details on the codes, see the Supplementary Material.

4 Findings

In this section, we present insights derived from participant responses, organized to align with our four research questions. Each subsection highlights findings directly based on the participants' inputs. To make these insights actionable for HCI researchers and tool designers, we follow each section with Key Insights that summarize why the findings are relevant for HCI researchers and enumerate areas for improvement and possible strategies for designing more user-friendly neuromorphic development tools. Figure 2 is the diagrammatic representation of the findings and its synthesis from the qualitative analysis of interview data.

Participant ID	Current Role	Qualification	Location	Programming Experience	Neuromorphic Experience
1	Post Doctoral Researcher	PhD Robotics	Canada	17	3
2	Senior Neuromorphic Engineer	Undergrad Nanotechnology Engineering	Canada	7	7
3	PhD Student	Masters Device Physics	US	5	2
4	PhD Student	Masters Embedded Systems	US	10	7
5	Post Doctoral Researcher	PhD Electronic and Computer Engineering	Australia	6	3
6	Lab Researcher	PhD Cognitive Science	Canada	24	11
7	PhD Student	Masters in Neural Systems and Computation	Germany	7	6
8	Undergrad Research Assistant	Undergrad Computer Engineering	US	6	1
9	Professor	PhD Neuromorphic Computing	US	10	8
10	Lab Researcher	PhD Cognitive Science	US	20	9
11	Lab Scientist	PhD Spiking Neural Networks	US	14	10
12	Lab Leader	PhD Computer Engineering	US	39	9
			Mean	13.75	6.33
			Median	10	7
			Max	39	11

Table 2: Background, location, and years of experience of the interview participants

4.1 Participants' Motivations towards Neuromorphic Computing

One of the primary motivators driving participants' interest in NC was the observable incompatibility of conventional computing models with real-time, on-board learning in robots due to the substantial power and computing requirements [P1]. Three participants [P1, P6, P12] emphasized the computational demand and resulting unattainable energy needs when using conventional AI in robotics and neuronal simulations.

The standard approach of integrating power-hungry processors on board or relying on tethered computational resources is also deemed untenable, as P1 analogized it with embedding a large power-hungry data center on a mobile robot, explaining its practical infeasibility. They stated that particularly in applications such as space and underwater robotics, which necessitate higher levels of autonomy and minimal power consumption, NC is better poised to meet the requirements of such systems. P1 also reported that transitioning from the Von Neumann architecture to a neuromorphic one wasn't as significant a shift for them and that the asynchronous nature of NC was seen as compatible with robotics' event-driven nature. They believed that the energy-saving prospects were substantial with NC.

While many participants focused on the energy efficiency of NC systems, P3 considered NC as the route to **recreating the unparalleled speed and accuracy of the human brain through technology**. They emphasized the promise embedded in SNNs to mirror neurons more truthfully than other neural network approaches:"*...replicating my brain on a chip or something like that, which is as fast as the brain, especially via neuromorphic...there is a possibility of being it closer to the normal biology, the biological principles*".

Neuromorphic accelerators, which can alleviate power constraints around running SNNs on traditional CPUs and other conventional hardware and allow for enhanced research capabilities in understanding neural networks and behaviors, were also of significant interest to our participants [P10, P12]. To take full advantage of the benefits NC has to offer, our participants [P4, P5, P9] collectively emphasized focusing on a "full-stack" or a highly interdisciplinary and collaborative approach to NC: *"I think in the end neuromorphic computing, it's the fullest stack computing... you'll see neuromorphic researchers that are mainly concerned with devices. Or you have physicists working on neuromorphic devices or materials. You have, for example, architecture, design, and algorithms. The ideal scenario is all of these will be together in one resource"*. (P9). This entails not only focusing on the software side but also considering the hardware aspects and bringing them together into one cohesive platform. A co-design mindset, where both hardware and software are developed in tandem, was deemed crucial to realize the benefits of NC.

Key Insights for HCI researchers: We believe that these perspectives from the participants converge towards a collective understanding that NC is not simply an exploration of an alternate computing paradigm but is fundamentally an iterative progression towards a model that embraces the efficiency and capability of the human brain. They point to the fact that NC affords unique features to systems like adaptability and fault tolerance, which present unique opportunities for HCI researchers to innovate in areas like user experience design and adaptive interfaces.

4.2 Existing Workflows, Tools, and Resources for Neuromorphic Development

When asked about the steps taken by developers to build neuromorphic applications, our participants shed light on the integrated workflow they use for neuromorphic system design, along with the tools and resources that they employ to create their applications.

4.2.1 Workflow. Our participants reported a workflow that was congruous to brainstorming [P1, P3, P5, P6, P8, P9, P10, P12], sandboxing/experimentation & debugging/validation [P1, P3, P5, P7, P8, P10, P11], followed by training an SNN [P2, P5, P8, P11, P12], optimizing the SNN [P1, P2, P5, P6, P7, P8, P11, P12], and finally moving the network from a simulator to neuromorphic hardware [P6, P12].

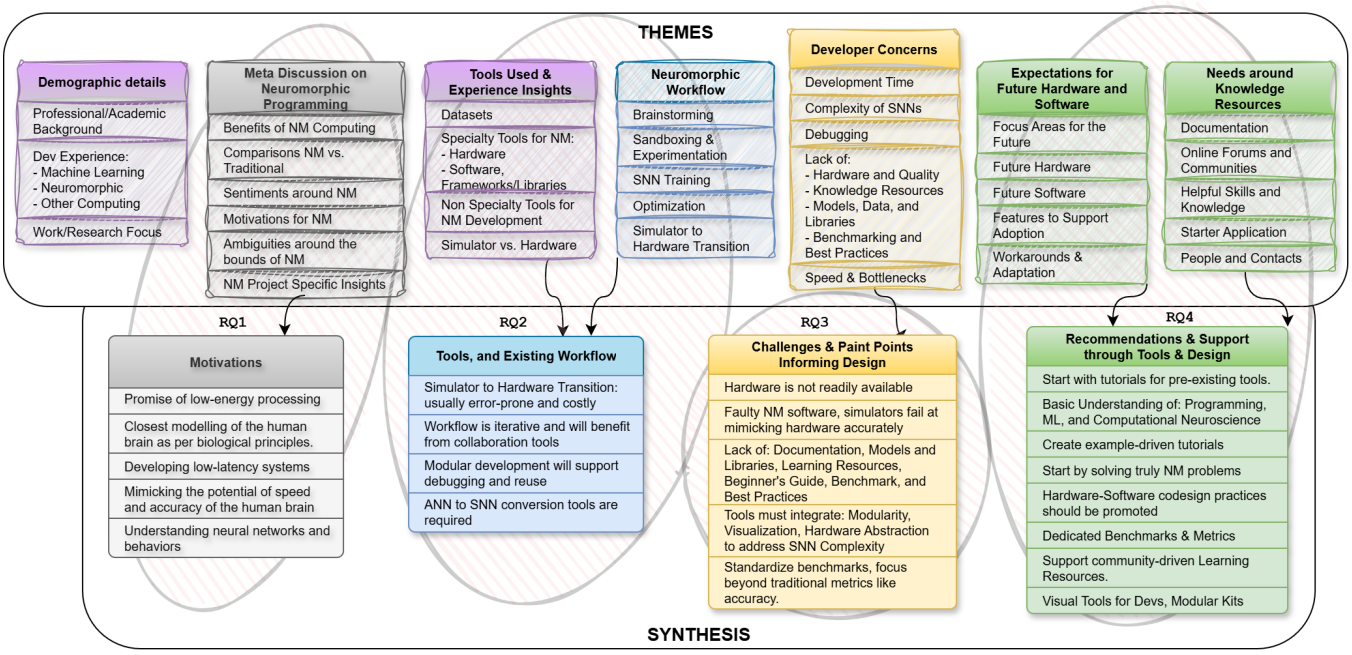


Figure 2: Mapping the emergence of themes from qualitative analysis codes. This diagram illustrates how codes derived from participant interviews were grouped into themes, showing their relationships to the key findings and discussions of the study.

Brainstorming begins with a clear problem definition [P8], and involves the exploration of the current architectural design and reusable code [P1, P2], creating solution designs with high-level blocks [P3], exploration and selection of tools and hardware [P8, P12], accounting for hardware and software constraints and compatibility [P6, P12], and creating flowcharts from unstructured ideas to create the solution structure [P5].

Sandboxing and Experimentation involves iterative programming, where participants emphasize the significance of reusing code from standard implementations [P1, P2], utilizing existing techniques such as scikit-learn and in-house code for baseline algorithms [P2], and incorporating tools like optimizers from SciPy [P2] depending on the development domain. Participants reported the usage of notebooks to create a sandbox for experimentation [P5, P10], which often involves probes and data collection to troubleshoot and debug spiking neural systems [P4]. The habit of building more modular networks is suggested for easier debugging [P4], but the participant acknowledges it as a practice that is still underutilized. Finally, validation can include either optimization of a trained SNN or optimization of a traditional ANN or other network, followed by conversion to a spiking version with aims to match the original model closely [P5].

The **SNN training** phase involves transitioning from a non-neuromorphic implementation to a spiking version, with complexities increasing as the model size grows [P2]. The participant also estimated that training SNNs takes up a significant portion (50-70%) of development time for projects. Troubleshooting involves adjusting parameters like firing thresholds. Citing the absence of widely accepted default workflows or a Stack Exchange-like platform, and noting the large knowledge gap between experts and novices, participants highlighted the need for clear communication and widely adopted channels for effective problem resolution between these groups [P6]. Training SNNs encompasses challenges associated with the lack of fundamental theory and the uncertainty of successful outcomes (i.e., whether data is learnable) and often involves an iterative process, making adjustments based on the results [P6, P7, P11].

Optimization strategies focus on making networks more efficient [P8]. Techniques such as making the networks more modular, selecting appropriate algorithms, and fine-tuning network size through hyperparameter optimization are employed [P8, P9]. Optimizing power consumption involves careful consideration of neuron placement and network abstraction [P10, P11].

Finally, the **simulator to hardware transition** is characterized by mapping algorithms from CPUs to neuromorphic hardware. Differential equations are used as a unique technique to express algorithms, facilitating their implementation on diverse neuromorphic hardware [P6]. Simulation on CPUs first is preferred, primarily because most tools are on traditional CPUs, and efforts are made to ensure seamless transitions to neuromorphic hardware with minimal code changes [P6, P12].

To **aid the conversion of non-neuromorphic code to neuromorphic SNN-based versions** in cases where the SNN is built through conversion from an ANN rather than from scratch, P3 and P4 reported a need for tools that could allow known traditional neural network models to be run directly on neuromorphic hardware. Despite the common expert sentiment about the necessary

demarcation in the scope and value proposition of neuromorphic and traditional computing, P3 and P4 reported that such conversion tools can help as a starting point for beginners with a good understanding of traditional ANNs. P3 also stated that it would be good to have*"some other tool-chains or frameworks... that are completely open so that and you give an ANN, and then you can completely get the SNN processed and get the output out. That complete framework would be extremely advantageous for users starting out".*

4.2.2 Tools and Resources. Participants reported on their usage of datasets, and various tools for neuromorphic development. [P5, P7, P11, P12] mentioned using datasets like the DVS gesture dataset [7], MNIST [42], and the Oxford Radcliffe dataset. They hinted at generic speech, video, and image datasets being used to learn Deep Learning basics, and how traditional Deep Learning techniques are already good at solving problems therein. In addition to datasets, 3 tabulates the tools used for programming in NC as reported by each of the participants.

Key Insights: For HCI researchers, understanding the detailed workflow and tools used in neuromorphic development is crucial for identifying key stages where user interfaces and experiences can be optimized to enhance productivity and reduce cognitive load. For example, knowing that developers prioritize modularity during the debugging phase can lead to the design of interfaces that better support component-based development and visualization tools that make complex networks more comprehensible. Furthermore, insight into the iterative nature of training SNNs helps HCI researchers create more effective collaborative tools that facilitate communication and knowledge-sharing between novices and experts to tackle the steep learning curve associated with SNNs.

4.3 Challenges and Pain Points in Neuromorphic Development

To better support HCI researchers in identifying areas for intervention, we asked participants about specific challenges they encounter in developing neuromorphic systems. In this section, we discuss our findings related to the challenges of neuromorphic programming.

4.3.1 Complexity of SNNs. A majority of the participants [P1-P5, P10-12] reported that the complexity of SNNs is something that most programmers struggle with. These challenges arise due to the usage of spikes as the primary signal, and the introduction of the **time dimension**.

Traditional ANNs, when implemented on a Von Neumann architecture, operate in discrete time cycles that are synchronized with a clock signal. Therefore, computations in these networks occur at fixed time intervals, typically driven by the clock speed of the CPU, when neurons both process their inputs and update their analog states. Instead, SNNs are asynchronous, incorporating time as an explicit dimension, allowing them to model and simulate the behavior of biological neurons more accurately. Drawing from biology, neurons in SNNs integrate their inputs continuously over time into their membrane voltage, and communicate using asynchronous spikes.

A spike refers to the all-or-none firing of a neuronal action potential whenever the membrane voltage exceeds a threshold. The asynchronous spike emission from each neuron complicates the relationship between input and output spikes and necessitates the inclusion of a temporal dimension and processing delays in a single neuron. The effect becomes aggravated in cascaded architectures, where the delays are propagated through the neuronal layers. Learning to work with spikes as the primary neuronal signal challenges the developers learning to program in the neuromorphic domain, with half of the participants [P1-P5, P11] reporting that this concept makes training or simulating SNNs difficult, as it is not directly translatable from ANNs. P2 states "*I guess the main thing is that it introduces delay, you have to learn to account for in some way...* ".

The sequence of multiple neuronal firing timings constitutes a spike train. Information from sensors needs to first be encoded into spike trains, which are then processed through the SNN which outputs spikes that are decoded to generate comprehensible information. Many participants [P2, P5, P8] reported difficulty in encoding data and interfacing the neural network with the neural processor in spikes to ensure they convey information accurately. They reported that the encoding process is an integral aspect of working with SNNs and the various available encoding techniques result in varying levels of accuracy and efficiency. Therefore, choosing the correct technique is both critical and challenging. P10 states "*...This is probably the first piece of advice that I give to people when they're struggling with projects is that they should look more carefully at how they're encoding information because it's very common to see others struggling with spike representations that are useless. And that leads to poor, poor performance and endless retraining*". P2 also states that "*...It also starts adding in just more knobs to tune so like, things like firing rate, how quickly should they, like, what ranges should they spike in?*".

The additional temporal dimension and the required encoding of continuous information into discrete spike trains **increases the number of hyperparameters** in SNNs. Hyperparameters in a neural network are variables like learning rate, batch size, etc., and their tuning is an essential process in achieving optimal performance. Four participants [P5, P7, P10, P12] raised concerns over the abundance of the hyperparameters that need to be tuned with SNNs. As P5 states "*...When I'm doing hyperparameter sweeps, I spend a lot more time on an SNN over a conventional Neural Network, because yep, more things to play with... thresholds, ...various types of batch normalization, ...a lot of degrees of freedom that I'm not sure will make my application better. So yeah, maybe I'll run 200 trials for a DL algorithm and be satisfied but with an SNN, that might end up being 600 trials... I haven't really done a direct one-to-one benchmark, but it could easily be five times the time on like a miscellaneous dataset*".

Key Insights: One of the ways in which the challenge of complexities within NC systems may be alleviated is through **designing modular and reconfigurable systems**. Three participants [P7, P9, P11] reported that being able to abstract neural network modules as much as possible and implement them easily without having to understand them from inside out would be better, especially for exploration or demonstration purposes. NC can benefit from drawing inspiration from current ML frameworks like TensorFlow or PyTorch that support such modular networks which can be used by beginners to explore their effectiveness. P1 states: "*The modularity in the system, making that easier... making it easier to run modules independently would be good. But that's not something that's precluded*

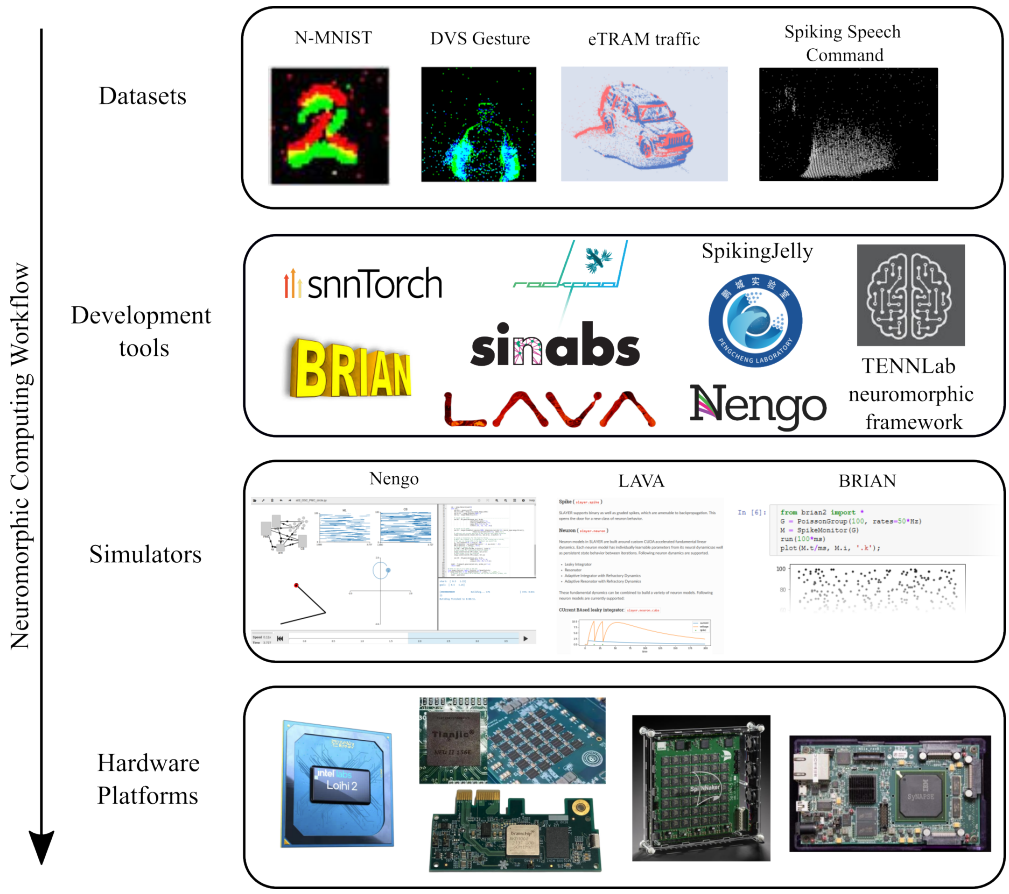


Figure 3: A standard workflow for neuromorphic computing applications. Starting from event-based vision and audio datasets, developers utilize custom tools and libraries to program their SNN-based algorithms. Simulators are used to emulate the performance of the developed algorithms before their deployment on neuromorphic hardware.

by Nengo, or Lava. It's more of a programming practice that I'm still developing with these models... I'd like to see a tool where... you can latch on to the model and get that (model functionality) without specifying it explicitly in the code itself... I think that would be the that would be the number one on my wish list if I could get that".

Considering the difficulty in tuning SNNs, HCI could contribute to developing more interactive, feedback-oriented tools that assist developers in understanding the effects of hyperparameter tuning in real-time.

Finally, being able to **clarify the value proposition for NC** would significantly simplify the decision-making process for new developers who're willing to explore NC for their applications. Four participants [P5, P6, P9, P11] reported that it is important to clearly establish the list of domains in which NC shows particular promise for greater performance and applicability compared to traditional computing. P6 reported that having a graphic that highlights the benefits of neuromorphic for some specific uses, if made ubiquitous in the neuromorphic community, would make it clear for early developers in the field. P11 reported that there should be a clear explanation of why neuromorphic excels on such domains and shows promise for greater performance than current technology. Without this distinction, a lot of early development effort in learning neuromorphic can be misguided, and developers would merely be copying traditional deep learning problems in neuromorphic.

For developers who are more experienced with traditional ML/AI workflows, **mirroring development flows and patterns of successful non-neuromorphic tools** could be another strategy to ease the transition to implementing neuromorphic systems. Several participants [P1, P2, P5, P7, P9, P11] agreed that when neuromorphic tools are similar in look, feel, and functionality to more traditional developer tools, it is easier for curious developers to try, test, and switch to neuromorphic systems. For example, P9 discussed how libraries for NC should be more similar to the libraries for popular ML and DL platforms with which more developers are already familiar: "*So for example, with Pytorch and SNNtorch, that is kind of like bringing in something that people are comfortable with, I don't want us to be so exotic that people are afraid of learning*". Programming practices such as abstraction, code modularity, standard compilers, and debugging strategies that currently exist in traditional programming are helpful and can be applied for neuromorphic programming as well [P1, P3, P4, P5, P9].

Participant	Speciality Development Tools	Hardware Tools
1	Nengo (Visualization), Intel's LAVA framework (Mapping and Programming)	Intel Loihi (NM Processor)
2	ABR neuromorphic compiler (Nengo), pytorch, TensorFlow, Nengo DL (Libraries and Frameworks), SNN toolbox (Multi-functional)	-
3	-	VHDL, Verilog, Vivado (Hardware Description Languages)
4	SNN toolbox, SNNTorch (Libraries and Frameworks)	Loihi, SpiNNaker, SVP (SpiNNaker Virtual Plasticity) (Programmable Hardware Platforms)
5	SNNTorch, PyTorch, Spiking Jelly, Intel's LAVA DL (Libraries and Frameworks), Nengo (Visualization)	SpiNNaker, Python, Embedded C, Loihi
6	Nengo, Nengo IDE (University of Waterloo), Lava, NX SDK	-
7	Lava, Brain 2 (Simulator)	-
8	TENNLab neuromorphic framework (C++), Lava, Pytorch, SNNtorch	-
9	TENN Lab framework, Pytorch, SNNtorch	Loihi, Lava framework
10	Slayer (Library), NVIDIA's reference architecture	Loihi
11	Slayer (Library), Lava, Loihi (Multi-functional)	BrainScale, DYNAPS (Hardware Platforms)
12	EON simulator, CrossSim (Simulators)	Intel Loihi, TrueNorth, FPGA implementation (Hardware Simulation)

Table 3: Categorized Hardware and Development Tool usage as reported by the participants.

4.3.2 Buggy Simulator Tools and Unavailability of Hardware. Neuromorphic simulators are meant to mimic neuromorphic hardware, thereby aiding experimentation, helping developers understand network interactions and neuron behavior, and accelerating the development of neuromorphic systems. However, many participants highlighted that **existing simulators and development tools are buggy** [P5, P7] and their APIs suffer from poor usability, particularly for beginners [P3]. P7 attributed this to a goal-oriented development process focused on specific hardware, neglecting beginner users.

Several participants [P1, P2, P3, P6, P7, P9, P10] reported difficulty with the use of simulators and transitioning from the simulator to hardware and vice-versa. As P3 reported, simulators only describe the hardware and results obtained on simulators are 'never' replicated on hardware. P6 reported that many **current simulators do not simulate hardware constraints** like the limit of neurons that can be supported by the hardware accurately, which causes failures in code. P7 agreed, reporting that while it is sometimes better to not have to be mindful of hardware constraints, it can cause issues when transferring the simulated models onto the hardware. P1 and P11 reported that the I/O speeds are generally slower in simulators than hardware.

Half of our participants [P1, P2, P4, P5, P6, P12] agreed that there is a **lack of available neuromorphic hardware**. This poses a challenge for neuromorphic development, as it often requires developers to switch back and forth between simulators and real hardware, increasing the time spent on and complexity of the task. P4 states "*...if we want to have actual neuromorphic hardware, there are not many companies that actually sell such systems readily available on the market. And yeah, I think that's the biggest resource constraint that we have.*". P6 and P12 agreed that the lack of readily available hardware impedes algorithm development, and can dissuade novice enthusiasts of NC from entering the field.

Access to NC hardware is usually limited to large research facilities, or companies that can afford the resources to enable access to the hardware. P12 also reported that there is a lack of open-source neuromorphic devices, where anyone seeking a device faces further red tape of committing to research and usage agreements, adding to the logistical challenges for a neuromorphic developer.

Additionally, P5 and P6 reported that some chips that were developed in an academic setting were available but are limited in scope. P5 felt that 'academic chips' are not necessarily designed with the broader community in mind. When hardware is available, it often requires significant hardware debugging. P10 reported having to work with Jupyter Notebook instead of their preferred development environment VS Code, and having to set up proper configurations for each remote layer, resulting in a slower workflow when compared to working with a local IDE.

4.3.3 Lack of Documentation, Models, Libraries, Best Practices and Benchmarks. All the participants unequivocally expressed that there is a **lack of knowledge resources and proper documentation making it difficult to find pertinent information regarding algorithmic development in NC and increasing the barrier to entry for novices**. Participants [P1, P6, P7, P10] reported that there is no online question-and-answer platform that programmers can refer to when they encounter unexplained errors that contain information about NC specific problems, such as Stack Overflow, requiring them to have to answer many questions by themselves. This increases developers' learning time and their overall development time.

Because the community of developers is small, the focus is primarily on implementation, not documentation and pedagogy. Participants [P1, P2, P5, P7] reported that the documentation for existing hardware and development tools is not as good as they would expect. P5 felt that the highly skilled researchers working on the development of new software packages and hardware had insufficient incentives to be creating robust documentation and P7 felt that this dearth of resources was due to a lack of financial investment in NC. P1 states "*There's not like a Stack Exchange for NC. Which is something I'm mildly worried about. Because I'm probably not going to be in this lab forever (in losing regular contact and access to experts)*".

Similarly, participants [P5, P7, P10] reported that the **currently available tutorials are inadequate**, with individuals moving into the neuromorphic field relying heavily on just the few available tutorials. P5 attributed the relative success of SNNTorch to the tutorials that it has, but reported that, although beginners go through the tutorials, the information they retain is low. With regard to sample starter applications, P7 and P10 reported that the tutorials are not available and when they are, the right kind of tutorials are difficult to find, especially for applications that are not from major vendors. While scholarly articles can serve as knowledge resources, participants [P5, P9, P10, P12] reported that there are few such articles and that they are usually complicated, too specific, or do not provide holistic solutions. P10 states "*we probably have, like, 1%, as many papers out there. And the papers that are out there, most of the good ones are written by neuroscientists not by programmers, or, you know, computer scientists. And so they tend to be more esoteric*".

Participants [P3, P10] noted that **models are scarce, hard to select based on requirements, and challenging to implement and optimize**. P3 mentioned that developing neuromorphic applications with languages like Verilog or Matlab requires building models from scratch, along with new documentation for simulation. This challenge also applies to encoding-decoding techniques, as developers must learn these anew for each system. Model development from scratch is also necessary when testing hardware for training SNNs, since pre-existing models are typically unavailable [P3]. While tools for ANN to SNN conversion exist [11, 29], P3 pointed out that the converted models aren't always compatible with all hardware languages, adding more complexity. In contrast, building ANNs in Python is much simpler due to the availability of pre-built models.

The repetitive building from scratch is due to a **lack of standard practices in neuromorphic software development**, as noted by multiple participants [P2, P8]. Unlike the established workflows in deep learning for key steps like design, encoding, training, and testing, neuromorphic development varies by hardware, as for example each developer has their own signal processing approach to perform encoding. P2 mentioned that much of the development is novel, with no standard references available. Similarly, P8 reported little to no standardization in signal types, encoding, or simulation techniques, making knowledge from one system often non-transferable to another. In traditional programming, libraries allow developers to reuse code and save time, but P6 noted that available NC libraries are relatively new and mainly offer basic functions like min() and max(). While non-NC-specific libraries can sometimes be used, there is insufficient support for event-based data processing, which is a key aspect of NC. This lack of standardization complicates comparing implementations and hinders the creation of broader standards for neuromorphic programming. A notable exception is Lava by Intel, an NC library recently developed to provide standard operations similar to those in traditional computing.

Participants [P6, P8] reported a **lack of specialized tools interfacing with neuromorphic hardware** while supporting network visualization, simulation, and optimization. Though neuromorphic algorithm development can start on standard tools like VS Code, such tools lack the necessary hardware optimization and visualization features. P6 emphasized the importance of GUIs and visualizers for research, noting that building such tools is a significant software challenge. P8 added that no widely accepted UIs exist, as each hardware platform uses its own tools. Aside from Nengo[11], most tools lack real-time network visualization, which helps developers debug, optimize, and assess neural networks. P6 also shared that the absence of proper visualization tools significantly delayed their progress.

Interfacing challenges extend beyond development tools to the limited interchangeability of programming languages. Unlike machine learning, where Python-based frameworks are widely used, neuromorphic applications are developed in various languages, requiring support across them. However, P8 noted that there is **no standard neuromorphic framework supporting multiple languages**, since most frameworks are designed for specific hardware from universities or research labs and are not cross-platform. As previously mentioned, each framework has its own workflow and practices, making it difficult to accommodate the full variety of systems.

Participants also reported a **lack of established benchmarks for neuromorphic systems**, including a lack of appropriate datasets, evaluation metrics, and system level benchmarks that account for the deep coupling between NC hardware and software implementations. This makes comparing different neuromorphic implementations difficult and does not allow NC implementations to be compared to the existing state-of-the-art solutions in DL. Participants [P2, P5, P8, P12] believe this lack of benchmarking is because NC is still incipient and has a relatively small community. The newness and small community also make the field of NC prone to rapid change. Developers find it increasingly challenging to adapt to these swift changes in concepts due to the lack of robust benchmarks. P5 says "*what is a good metric of success, what is the most commonly accepted way to achieve X, Y, and Z?... We're all just figuring it out. It's such a volatile field. Things are changing all the time... Quite often things that we believe are really good, turn out to just be reimplementation of things that already existed in the past*".

In deep learning, there are standardized datasets for evaluating algorithmic performance such as ImageNet [26] for image classification tasks or CIFAR-10 [40] for image generation tasks. These datasets can give developers a better sense of their algorithm's performance and can show which techniques make their models more efficient, learn faster, or increase stability. P12 reported a **lack of standards for NC datasets**, claiming that pairing the right dataset with the right model can be difficult due to the lack of established guidelines.

Finally, **another crucial challenge lies in selecting appropriate evaluation metrics for NC**. DL benchmarks often prioritize accuracy, making them unsuitable for fully assessing neuromorphic algorithms, which may offer benefits such as low latency, low power consumption, or minimal footprint with marginal impact on accuracy. Nevertheless, both P5 and P9 noted that accuracy remains the primary metric for evaluating spiking neural network (SNN) implementations. Additionally, applying conventional DL benchmarks to traditional deep learning issues undermines the broader scope and unique aspects of NC [P9]. P9 further argues that metrics derived directly from DL performance can differ considerably from those relevant to NC. P5 also mentions that appropriate performance metrics for NC models should be closely aligned with the specific problem being addressed. For instance, using neuromorphic architectures for object detection may result in the lowest power when paired with event-based cameras rather than RGB cameras, rendering datasets like ImageNet ineffective for fully evaluating neuromorphic solutions. However, because NC is a comparatively small field, it is often expected to be evaluated against existing deep learning algorithms using existing datasets and benchmarks, posing a significant challenge for the adoption of neuromorphic solutions. This underscores the necessity for distinct system-level benchmarks that consider the full stack of hardware and software solutions.

Unfortunately, **traditional benchmarks are not well-suited for assessing neuromorphic systems**. Even edge solution benchmarks like MLPerf, which evaluate latency and energy as well as algorithmic accuracy, might be inadequate for neuromorphic solutions. Two participants [P5, P9] highlighted the challenge of estimating power requirements and selecting from different performance metrics, recommending that additional metrics be included in evaluations. P9 also proposed that the neuromorphic community should focus on different metrics like resilience, energy efficiency, and latency. However, they acknowledged that integrating design and performance metrics into a single framework is extremely challenging, as defining and selecting the most relevant metrics for a NC application or algorithm is not straightforward.

Key Insights: Based on our participants' suggestions, it is evident that a coordinated effort to create and maintain dedicated online platforms, coupled with incentives for active participation, could transform the NC knowledge resources landscape. Such platforms would not only serve as knowledge hubs but also strengthen the sense of community, fostering collaboration and collective problem-solving. Aside from these, **creating and incentivizing good documentation** with information at all levels of abstraction will empower developers to use the information for lower-level development such as programming neurons to higher-level usage such as adopting an API into their development pipeline [P5, P7].

Focusing on pedagogical materials such as example-driven tutorials [P1, P10] and the need for user friendly APIs [P3] would also serve as useful strategies in increasing knowledge dissemination for novice NC developers. Additionally, **creating comprehensive "Getting Started" resources** [P2, P11] that provide accessible introductory resources for beginners to navigate the intricacies and establish foundational knowledge was deemed useful. "*It'd be nice to have something like... Andrew Ng's machine learning journey, I think it's a very short document. And it goes through a big list of like tips if you're doing the typical back prop or a conventional machine learning. It'd be nice to have a version of that for like the neuromorphic implementations, the tips and tricks for if you're facing this type of problem, or like what type of neurons might be good for certain problems*" (P2).

To ensure that beginners have a clear learning pathway, **GUIs for beginners and real time network visualizers** could enable an abstract understanding of the network and the spiking activities occurring within the neurons, and help debug issues. This guideline is supported by many of our participants [P2, P5, P6, P8, P10, P11, P12], with P5 stating: *"...the moment you start using GUIs, and visualizations that kind of abstracts away the lower level detail that we want to play with. But for beginners being inducted into the field, I think it's fantastic."*

Furthermore, **having known best practices and guidelines for developers** with information such as what is the right workflow for developing a particular kind of neuromorphic application, what is the right encoding method for a particular type of data, what performance metric is suited for a particular problem, and what kind of dataset should be used to benchmark a particular type of algorithm would benefit the community at large [P5]. This is necessary so that developers can fall back on established best-practices for key decisions instead of having to invent and create new solutions for each challenge they face. This also allows the community to better evaluate the progress made on algorithm development when its performance can be directly compared to other algorithms.

The lack of standard benchmarks and practices also presents an opportunity for HCI designers to facilitate community-driven platforms that encourage sharing and standardization of best practices and resources. More pointed guidelines on how the HCI community can support a better development ecosystem for NC by taking inspiration from what's already been done in the past for other fields are discussed in section 5.

4.4 Suggested Prerequisites and Resources for Getting Started with Neuromorphic Computing

In this section, we present findings related to skills and resources that our participants thought could benefit novice developers in NC. These emerged from participants' responses regarding knowledge prerequisites and starter applications that would be interesting for newcomers.

4.4.1 Starter Applications for New Developers. The vastness of computational neuroscience knowledge, along with the lack of knowledge resources for understanding the neuromorphic domain make entry to the field challenging for beginners. One approach is to create starter applications as entry points to the field [17, 67]. A fourth of our participants acknowledged that **starting with tutorials for pre-existing tools is helpful for learning the basics of neuromorphic programming**. P1 and P5 mentioned Nengo [56] and P11 mentioned Intel's Lava [23] tutorials and the associated visualizations as having the right amount of complexity for beginners, with just enough abstraction of the neural processing and progressive disclosure of necessary information.

Most participants [P2, P3, P5, P6, P8, P10] mentioned that an effective introduction to the field could be starting with less complex problems that allow for fast model training and using datasets suited to the neuromorphic domain. Participants [P4, P5, P7, P10, P12] emphasized that new developers should **focus on problems where NC offers clear advantages, such as speed and power efficiency, over non-spiking implementations**. Several participants [P6, P8, P9, P10] suggested examples such as adaptive control for robotics or other small robotics application. P12 suggested that solving such problems with open-source tools like Nengo [11] and simulators like Brian [32] and Nest [31] is a good way to start learning.

If novices opt to start with a classic DL problem, P4 suggested digit recognition using the MNIST dataset as a good starting point and P11 further suggested transitioning to analogous neuromorphic tasks such as the Oxford Spike Raster translation. However, P7 noted that while approaching this well-studied problem can help with understanding, learning for truly neuromorphic solutions should focus on neuromorphic-directed event-based inputs, and **utilize visualizations of the neuronal spiking activities such as raster plots** that help the learner understand the complex dynamics of neurons like firing patterns, stimulation response, and more.

4.4.2 Helpful Skills. Participants reported several skills that could be considered as prerequisites to learning programming for neuromorphic systems.

Four participants reported that it is **beneficial to begin learning neuromorphic programming after understanding the basics of ML and programming** [P1, P2, P5, P7]. Based on their personal experience, P2 reported: "*I guess one of the earlier issues I had was I started the field not knowing much about neural networks. So I was kind of learning it from that end (NC)... So that was a bit of a challenge in the beginning. So I definitely think it's nice having a bit of a ML background to kind of get the basics and then starting to build up how this (NC) is different.*" Two participants [P5, P6] further reported that a strong programming background simplifies the transition to neuromorphic programming. Several concepts like sandboxing, simulation, optimization, and programming decisions translate from traditional to neuromorphic programming. P6 further reported that parallel (Java, C, and Python), and lower-level programming (VHDL, Verilog, and Simulink) are much closer to neuromorphic programming.

Six participants [P4, P5, P6, P9, P11, P12] stressed the importance of **building familiarity with neuromorphic hardware** and understanding how neurons and synapses are implemented on-chip in order to more effectively harness the efficiency of the hardware. P11 specifically states "*understanding the architecture, it's one of the keys, and also like here, and that if you don't, fully understand the limitation and how the systems work, I mean, you kind of will not be able to tell like, if you can implement certain things on the chip itself, Right? And that is critical*".

Four participants [P5, P7, P9, P12] reported that having a **basic background in computational neuroscience**, the field that utilizes mathematical models to understand how the brain works, is helpful, especially as a beginner. However, the field is dense and programmers can benefit from some abstraction [P12]. Specifically, P12 states "*I think the most important thing, at least for me,... is to get an intuition of how neurons can compute, how brains can compute it all. Right, it's really on that intuitive level, how can a network of neurons implement complex computations, as we observe, like in us and in animals, right, does this for me, the most important thing that I kind of learned*". Four participants [P4, P6, P8, P9] suggest that the combination of skills developed through both computer science and computational neuroscience makes for an easier transition to neuromorphic programming and excelling therein.

Three participants [P1, P2, P8] suggested better programming practices that could help with learning neuromorphic programming. P1 and P2 recommended **building modular networks that can be used abstractly**. Specifically, P1 states "*getting into the habit of building modular, more modular networks, I usually just build, you know, kind of small single use networks for exploration, or demonstration purposes*". P8 reported that learning to leverage preexisting documentation from other developers and writing documentation are also important skills.

Key Insights: For HCI researchers, these insights point to areas where educational tools and interfaces can be improved to support NC learning. One potential approach is creating adaptive learning environments that adjust content and complexity based on a learner's background in machine learning and programming. These environments could feature interactive, problem-solving simulations that gradually introduce more complex neuromorphic concepts, such as spike-based processing or neuromorphic chip architectures. Enhancing documentation tools to support a detailed understanding of neuromorphic projects could ease the steep learning curve. Finally, well documented maker kits that are inexpensive to acquire and test on could also prove to be a useful tool for novice developers in the area, and is an area that HCI researchers are well versed in. By designing interfaces and platforms that address these educational needs, HCI researchers can make NC more accessible and foster a larger, more skilled community of developers in this emerging field.

5 Leveraging Insights from Developer Tools in Traditional Computing to Inform Future Directions for Neuromorphic Development

While Section 4 presents the findings from participant interviews and key insights relevant to HCI researchers, this section builds upon those insights to offer actionable recommendations for designing better tools and systems for neuromorphic computing. Drawing on lessons from adjacent fields, such as machine learning and deep learning, we identify specific practices and approaches that can inspire improvements in the neuromorphic development ecosystem. This section is particularly relevant for HCI tool designers seeking to create accessible, intuitive, and community-driven platforms that address the challenges outlined in Section 4. By connecting the findings to established successes in traditional computing domains, we aim to provide a roadmap for advancing the developer experience in neuromorphic computing.

With the first challenges arising when novices attempt to learn about NC, the scarcity of knowledge resources accentuates the need for more, better-structured, and increasingly in-depth tutorials. The

established fields of ML and DL not only have an abundance of such resources, but can also lend tools like Torii and Sodalite that have been developed to **simplify and optimize the development of structured curriculum and documentation** [35, 37]. By encouraging experienced neuromorphic developers to adopt tools like these, we can ensure more of high-quality tutorials and documentation. In addressing the questions that arise during the solitary exploration of existing knowledge resources, novices to the field can benefit from platforms where community members exchange acquired knowledge and provide support. Knowledge forums like StackOverflow [9] available for conventional software development and DL have accelerated the identification and solution of problems and could similarly benefit an emergent field such as NC. In prioritizing the creation of a StackExchange community and **supporting more community-driven initiatives**, we can ensure new developers have the resources required to get their questions answered. Lastly, when new developers mature enough to dive into existing source code, the lack of detailed justifications on the design decisions and large, complex codebases is disheartening. This is exactly where the adaptation of information foraging tools like Wandercode and RIT [36, 53] can **provide recommendations about the sequence of attention points in developed code** and simplify the understanding and learning experience. Taking it one step further, assistive tools that have been developed for code analysis such as Tutorons and Ivie [34, 73] can be of help by **providing more verbose and user-friendly line-by-line code explanations**. Ensuring these tools can give valuable suggestions and guidance in the field of NC ensures new developers have all the learning resources required to develop the skill-set and confidence required to move on to developing SNNs of their own. Adapting these tools to NC however, is not always a trivial task. Certain software, like documentation or information foraging tools, tend to be abstracted away from the underlying codebase and are easier to utilize, simply requiring adoption into a developer's workflow. Other tools that rely more on domain-specific knowledge, like Tutorons, would require expert adaptation to be useful in the NC space but Ivie proposes an interesting potential solution to this, using LLMs to automatically generate explanations. Given the relative newness and lack of neuromorphic documentation, these explanations would need to be evaluated for specificity and accuracy before full adoption of the tool.

Stepping into action for the first time with specific problems in hand, new developers find themselves overwhelmed by the multitude of datasets and the previously proposed network models in the context of NC. The more mature field of DL has already faced and addressed these challenges: First, the use of well-established dataset repositories like Kaggle or Torchvision Datasets [45] has simplified the packaging, distribution, and standardization of datasets for a wide range of tasks. Similar efforts have started in the field of NC [43] where **conventional datasets need to be extended with their event-based counterparts**, but their adoption would need to become wider. Second, in the context of DL, several sophisticated tools have been developed to **facilitate the exploration and selection of network architectures based on the given task** [69, 72]. Tools like ExampleNet [72] utilize the collective knowledge of previously suggested effective models to provide architecture recommendations that range from network type to number and width of layers, and from neuron activations to ranges of hyperparameter values. Utilizing such tools in the context of NC and expanding them with the specifics of SNNs (e.g. spiking neuron models, input encoding, etc.) could provide a valuable starting base for new developers in the field.

As developers progress from foundational learning to more advanced neuromorphic development, the need for detailed, specialized tools becomes increasingly essential. For instance, dedicated DL tools take the initial hyperparameter recommendations discussed above and perform parameter sweeps or grid searches to identify their optimal values [3, 14, 22, 65]. Adapting similar tools to the NC context could significantly **aid developers in fine-tuning SNNs without excessive guesswork**. The domain transfer would be relatively straightforward as several parameters (learning rate, batch size, etc.) translate directly from DL to NC. For others, the tools should be extended to handle the larger parameter sets in NC, while considering their inter-dependencies (e.g. membrane capacitance affects the maximum neuronal firing rate given a number of time steps, with all three are seemingly independent parameters). When such parameter sweeps are conducted, however, bugs often emerge, underscoring the importance of **architecture-specific debugging tools**. DL already offers solutions, like UMLAUT [57], that provide architecture-specific debugging recommendations. Such tools could be extended to neuromorphic development, to **pinpoint specific sources of bugs in the heavily parameterized SNNs**. In addition to understanding network architectures, gaining insights into the inner workings of networks through real-time visualization tools can provide a more intuitive grasp of the network's behavior, especially considering the time-dependent nature of spike-based computations. Tools that allow for **real-time probing of certain parameters during the training phase** [61] can have a significant impact on the model accuracy, but must be adapted to suit the temporal dimension of SNNs. Such frameworks developed for DL optimization can be adapted and extended for NC, helping to streamline model refinement and boost overall performance.

Unlike conventional DL, accuracy is just one of several key metrics used in NC to evaluate a model's performance. Therefore, profiling a model based on memory allocation, power consumption, and latency becomes essential. While there are tools in place to evaluate ANNs for some of these metrics [78], adapting them to SNNs is crucial. This is especially important because performance evaluations for SNNs can differ significantly when run on actual neuromorphic processors compared to conventional hardware. As a result, accurate simulators are necessary to estimate SNN performance both on and off dedicated hardware. Though some simulators already exist [71], there is a growing need for **cross-platform systems that enable meaningful comparisons across different hardware**. Additionally, once deployed to hardware, better tooling like Inline [13] are needed to monitor these metrics. These efforts align with the ongoing push in the NC community for advanced benchmarking [77], addressing both software and hardware performance evaluations.

6 Discussion and Future Work

Our study highlighted that developers are motivated by the potential of Neuromorphic Computing (NC) to address critical needs in

low-power, real-time computing. However, they face significant challenges due to the nascent state of tools and resources available to support their workflows (RQ1). Through interviews with experts, we explored the workflows and tools currently used in NC development, emphasizing the need for more robust, modular, and standardized platforms to streamline practices and reduce fragmentation (RQ2). We also identified critical pain points, including steep learning curves, limited educational resources, and difficulty scaling solutions from simulation to hardware (RQ3). Additionally, we uncovered the skills, resources, and support structures necessary for new developers to onboard successfully into the NC ecosystem, highlighting the importance of comprehensive tutorials, pre-configured templates, and community-driven platforms (RQ4). These findings formed the foundation for the actionable recommendations detailed in Section 5, providing HCI researchers and tool designers with practical pathways to enhance the NC development ecosystem.

In recent years, the field of NC has seen significant advancements aimed at making the technology more accessible and developer-friendly, aligning closely with the findings of our study. One notable development is **NeuroBench, a framework designed for benchmarking NC algorithms and systems** that is collaborative and fair [76]. NeuroBench addresses the need for more standardized, flexible benchmarks, allowing for hardware-independent testing and offering a common open-source toolset to developers. By decoupling algorithmic performance from hardware and providing a structured benchmark suite, it encourages broader participation in neuromorphic research and prototyping, even for those without direct access to specialized hardware.

Progress has also been made in increasing hardware accessibility, with designers developing approaches to **implement SNNs on non-neuromorphic hardware**. New tools for FPGA-based neuromorphic implementation are providing a configurable, open-source spiking neural core architecture that can be deployed on readily available hardware. [21, 47]. Platforms such as Quantisenc and Spiker+ offer modular tools for developers to experiment with neuromorphic systems on familiar electronics. This architecture emphasizes flexibility, giving developers the ability to tailor SNNs for specific applications, without requiring deep knowledge of the underlying hardware. Other groups are taking accessibility a step further, such as efforts out of the University of Tennessee to create a neuromorphic starter kit, using only microcontrollers and common board computers to assemble neuromorphic systems [54]. The project aims to develop open-source electronics and software to enable deployment of neuromorphic designs by students and designers on familiar platforms such as Raspberry Pi, enabling real world prototyping and validation of SNN designs.

In addition to hardware-related advances, a suite of new software tools, including CARLsim++ [50], SNNAX [44], and GenericSNN [46] have emerged. These efforts aim to lower the entry barrier for neuromorphic programming by providing **user-friendly APIs, GUIs and simulators** while increasing compatibility with existing software tools, allowing developers to simulate and experiment with SNNs on conventional platforms and accelerating neuromorphic adoption beyond niche applications.

In the context of supporting broader adoption and bridging the interdisciplinary gaps identified in our research, these developments represent substantial progress. Where practitioners once faced hurdles such as limited hardware access, the community is now equipped with a growing array of tools that democratize access to neuromorphic technology. These advancements in neuromorphic accessibility also present an exciting opportunity for the CHI community. With the growing availability of neuromorphic tools, frameworks, and benchmarks, HCI researchers can now more easily engage with this evolving field. This opens up new possibilities for creating human-centered design solutions that can support broader adoption and practical application of NC in everyday systems.

Our study, however, focused primarily on interviews with experts to understand their motivations, workflows, and challenges. While this provided valuable insights, we acknowledge that observing participants in their natural workflows could further enrich our findings. Future research could adopt traditional contextual inquiry methods, such as observing participants as they use NC tools, identifying real-time pain points, and validating findings through direct feedback. Embedding researchers within participants' environments would offer a deeper understanding of tool usage and reveal hidden challenges, providing actionable insights for designing more effective and intuitive developer tools for NC.

7 Conclusion

Our paper identified critical challenges in neuromorphic programming, such as limited learning resources and hardware availability, fragmented community support, and difficulties in navigating complex codebases for new researchers. Drawing insights from neighboring fields like ML and DL, we proposed several approaches to address these issues, including better-structured tutorials, community-driven support platforms, and enhanced debugging and performance profiling tools. Recent developments in NC, such as standardized benchmarking frameworks and new developer libraries, are helping to lower barriers for entry, making it easier for both novice and experienced developers to engage with the field. By leveraging these advancements and creating accessible, developer-friendly tools, we can reduce cognitive barriers, fostering broader adoption and innovation in NC. Ultimately, the convergence of NC and HCI holds the potential to drive the creation of more intelligent, efficient, and scalable edge AI systems.

Acknowledgments

We sincerely thank Noah Pacik-Nelson for his valuable insights and tremendous contributions in refining the manuscript for our paper.

References

[1] [n. d.]. Event-Based Camera Chips Are Here, What's Next? - IEEE Spectrum. https://spectrum.ieee.org/event-based-camera-chips

[2] [n. d.]. Neuromorphic electronic systems | IEEE Journals & Magazine | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/58356

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals,

Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[4] Gregory D. Abowd. 2020. The Internet of Materials: A Vision for Computational Materials. *IEEE Pervasive Computing* 19, 2 (2020), 56–62. https://doi.org/10.1109/MPRV.2020.2982475

[5] Mohammed Riyaz Ahmed and B.K. Sujatha. 2015. A review on methods, issues and challenges in neuromorphic engineering. In *2015 International Conference on Communications and Signal Processing (ICCSP)*. 0899–0903. https://doi.org/10.1109/ICCSP.2015.7322626

[6] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: a case study. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice* (Montreal, Quebec, Canada) *(ICSE-SEIP '19)*. IEEE Press, 291–300. https://doi.org/10.1109/ICSE-SEIP.2019.00042

[7] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. 2017. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7243–7252.

[8] ATLAS.ti. 2023. *ATLAS.ti Scientific Software Development GmbH*. https://atlasti.com/ Version 9, Mac/Windows.

[9] Jeff Atwood and Joel Spolsky. 2008. StackOverflow. https://stackoverflow.com/

[10] Mark Barnell, Courtney Raymond, Lisa Loomis, Darrek Isereau, Daniel Brown, Francesca Vidal, and Steven Smiley. 2023. Advanced Ultra Low-Power Deep Learning Applications with Neuromorphic Computing. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–4. https://doi.org/10.1109/HPEC58863.2023.10363561 ISSN: 2643-1971.

[11] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C Stewart, Daniel Rasmussen, Xuan Choo, Aaron Russell Voelker, and Chris Eliasmith. 2014. Nengo: a Python tool for building large-scale functional brain models. *Frontiers in neuroinformatics* 7 (2014), 48.

[12] Gennadi Bersuker, Maribeth Mason, and Karen L Jones. 2018. Neuromorphic computing: The potential for high-performance processing in space. *Game Changer* (2018), 1–12.

[13] Andrea Bianchi, Zhi Lin Yap, Punn Lertjaturaphat, Austin Z. Henley, Kongpyung Justin Moon, and Yoonji Kim. [n. d.]. Inline Visualization and Manipulation of Real-Time Hardware Log for Supporting Debugging of Embedded Programs. 8 ([n. d.]), 248:1–248:26. Issue EICS. https://doi.org/10.1145/3660250

[14] Lukas Biewald. 2020. Experiment Tracking with Weights and Biases. https://www.wandb.com/ Software available from wandb.com.

[15] Peter Blouw and Chris Eliasmith. 2020. Event-driven signal processing with neuromorphic computing systems. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 8534–8538.

[16] Shekhar Borkar. 2012. Technology challenges and opportunities for ubiquitous computing. In *2012 IEEE Asian Solid State Circuits Conference (A-SSCC)*. 125–128. https://doi.org/10.1109/IPEC.2012.6522621

[17] Peter Brusilovsky, Lauri Malmi, Roya Hosseini, Julio Guerra, Teemu Sirkiä, and Kerttu Pollari-Malmi. 2018. An integrated practice system for learning programming in Python: design and evaluation. *Research and practice in technology enhanced learning* 13 (2018), 1–40.

[18] Kurt Cagle. [n. d.]. (1) Why The Future Of Computing Is Heterogeneous | LinkedIn. https://www.linkedin.com/pulse/why-future-computing-heterogeneous-kurt-cagle/

[19] Carrie J. Cai and Philip J. Guo. 2019. Software Developers Learning Machine Learning: Motivations, Hurdles, and Desires. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 25–34. https://doi.org/10.1109/VLHCC.2019.8818751

[20] Kristofor D Carlson, Jayram Moorkanikara Nageswaran, Nikil Dutt, and Jeffrey L Krichmar. 2014. An efficient automated parameter tuning framework for spiking neural networks. *Frontiers in neuroscience* 8 (2014), 10.

[21] Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo. 2024. Spiker+: a framework for the generation of efficient Spiking Neural Networks FPGA accelerators for inference at the edge. *arXiv preprint arXiv:2401.01141* (2024).

[22] Andrew Chen, Andy Chow, Aaron Davidson, Arjun DCunha, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Clemens Mewald, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Avesh Singh, Fen Xie, Matei Zaharia, Richard Zang, Juntai Zheng, and Corey Zumar. 2020. Developments in MLflow: A System to Accelerate the Machine Learning Lifecycle. In *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning* (Portland, OR, USA) *(DEEM '20)*. Association for Computing Machinery, New York, NY, USA, Article 5, 4 pages. https://doi.org/10.1145/3399579.3399867

[23] Intel Corporation. 2021. Lava: An Open-Source Neuromorphic Computing Framework. https://lava-nc.org/ Accessed: 2024-05-31.

[24] Shaveta Dargan, Munish Kumar, Maruthi Rohit Ayyagari, and Gulshan Kumar. 2020. A survey of deep learning and its applications: a new paradigm to machine learning. *Archives of Computational Methods in Engineering* 27 (2020), 1071–1092.

[25] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. 2018. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro* 38, 1 (2018), 82–99. https://doi.org/10.1109/MM.2018.112130359

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

[27] J. P. Dominguez-Morales, Angel Jiménez-Fernandez, Manuel J. Domínguez-Morales, and Gabriel Jiménez-Moreno. 2017. NAVIS: Neuromorphic Auditory VISualizer Tool. *Neurocomputing* 237 (2017), 418–422.

[28] Jason K. Eshraghian, Max Ward, Emre O. Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D. Lu. 2023. Training Spiking Neural Networks Using Lessons From Deep Learning. *Proc. IEEE* 111, 9 (2023), 1016–1054. https://doi.org/10.1109/JPROC.2023.3308088

[29] Wei Fang, Yanqi Chen, Jianhao Ding, Zhaofei Yu, Timothée Masquelier, Ding Chen, Liwei Huang, Huihui Zhou, Guoqi Li, and Yonghong Tian. 2023. SpikingJelly: An open-source machine learning infrastructure platform for spike-based intelligence. *Science Advances* 9, 40 (2023), eadi1480. https://doi.org/10.1126/sciadv.adi1480 arXiv:https://www.science.org/doi/pdf/10.1126/sciadv.adi1480

[30] Kai Gao, Zhixing Wang, Audris Mockus, and Minghui Zhou. 2023. On the Variability of Software Engineering Needs for Deep Learning: Stages, Trends, and Application Types. *IEEE Transactions on Software Engineering* 49, 2 (2023), 760–776. https://doi.org/10.1109/TSE.2022.3163576

[31] Marc-Oliver Gewaltig and Markus Diesmann. 2007. Nest (neural simulation tool). *Scholarpedia* 2, 4 (2007), 1430.

[32] Dan FM Goodman and Romain Brette. 2008. Brian: a simulator for spiking neural networks in python. *Frontiers in neuroinformatics* (2008), 5.

[33] Hananel Hazan, Daniel J Saunders, Hassaan Khan, Devdhar Patel, Darpan T Sanghavi, Hava T Siegelmann, and Robert Kozma. 2018. Bindsnet: A machine learning-oriented spiking neural networks library in python. *Frontiers in neuroinformatics* 12 (2018), 89.

[34] Andrew Head, Codanda Appachu, Marti A. Hearst, and Bjorn Hartmann. 2015. Tutorons: Generating context-relevant, on-demand explanations and demonstrations of online code. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (Atlanta, GA, 2015-10). IEEE, 3–12. https://doi.org/10.1109/VLHCC.2015.7356972

[35] Andrew Head, Jason Jiang, James Smith, Marti A. Hearst, and Björn Hartmann. [n. d.]. Composing Flexibly-Organized Step-by-Step Tutorials from Linked Source Code, Snippets, and Outputs. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2020-04-23) *(CHI '20)*. Association for Computing Machinery, 1–12. https://doi.org/10.1145/3313831.3376798

[36] Austin Z. Henley, David Shepherd, and Scott D. Fleming. [n. d.]. Wandercode: An Interaction Design for Code Recommenders to Reduce Information Overload, Ease Exploration, and Save Screen Space. arXiv:2408.14589 [cs] http://arxiv.org/abs/2408.14589

[37] Amber Horvath, Andrew Macvean, and Brad A. Myers. [n. d.]. Support for Long-Form Documentation Authoring and Maintenance. IEEE Computer Society, 109–114. https://doi.org/10.1109/VL-HCC57772.2023.00020

[38] Nikola Kasabov, Nathan Matthew Scott, Enmei Tu, Stefan Marks, Neelava Sengupta, Elisa Capecci, Muhaini Othman, Maryam Gholami Doborjeh, Norhanifah Murli, Reggio Hartono, et al. 2016. Evolving spatio-temporal data machines based on the NeuCube neuromorphic framework: Design methodology and selected applications. *Neural Networks* 78 (2016), 1–14.

[39] Thomas Kluyver, Benjain Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter Development Team. 2016. Jupyter Notebooks—a publishing format for reproducible computational workflows. In *IOS Press*. IOS Press, 87–90. https://doi.org/10.3233/978-1-61499-649-1-87

[40] A. Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images, CIFAR-10 Dataset. https://www.cs.toronto.edu/~kriz/cifar.html. Accessed: September 7, 2024.

[41] Thomas D. LaToza, Arturo Di Lecce, Fabio Ricci, W. Ben Towne, and André van der Hoek. 2019. Microtask Programming. *IEEE Transactions on Software Engineering* 45, 11 (2019), 1106–1124. https://doi.org/10.1109/TSE.2018.2823327

[42] Yann LeCun. 1998. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/* (1998).

[43] Gregor Lenz, Kenneth Chaney, Sumit Bam Shrestha, Omar Oubari, Serge Picaud, and Guido Zarrella. 2021. *Tonic: event-based datasets and transformations.* https://doi.org/10.5281/zenodo.5079802 Documentation available under https://tonic.readthedocs.io.

[44] Jamie Lohoff, Jan Finkbeiner, and Emre Neftci. 2024. SNNAX–Spiking Neural Networks in JAX. *arXiv preprint arXiv:2409.02842* (2024).

[45] Sébastien Marcel and Yann Rodriguez. 2010. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on*

Multimedia (Firenze, Italy) *(MM '10)*. Association for Computing Machinery, New York, NY, USA, 1485–1488. https://doi.org/10.1145/1873951.1874254

[46] Alberto Martin-Martin, Marta Verona-Almeida, Rubén Padial-Allué, Javier Mendez, Encarnación Castillo, and Luis Parrilla. 2024. GenericSNN: a framework for easy development of Spiking Neural Networks. *IEEE Access* (2024).

[47] Shadi Matinizadeh, Noah Pacik-Nelson, Ioannis Polykretis, Krupa Tishbi, Suman Kumar, M Lakshmi Varshika, Arghavan Mohammadhassani, Abhishek Mishra, Nagarajan Kandasamy, James Shackleford, et al. 2024. A Fully-Configurable Open-Source Software-Defined Digital Quantized Spiking Neural Core Architecture. *arXiv preprint arXiv:2404.02248* (2024).

[48] Don Monroe. 2014. Neuromorphic computing gets ready for the (really) big time. *Commun. ACM* 57, 6 (June 2014), 13–15. https://doi.org/10.1145/2601069

[49] Dylan R. Muir, Felix Bauer, and Philipp Weidel. 2019. Rockpool Documentaton. https://doi.org/10.5281/zenodo.3773845

[50] Lars Niedermeier and Jeffrey L Krichmar. 2024. An Integrated Toolbox for Creating Neuromorphic Edge Applications. *arXiv preprint arXiv:2404.08726* (2024).

[51] University of Manchester. [n. d.]. SpiNNaker: A Neuromorphic Computing Platform. http://apt.cs.manchester.ac.uk/projects/SpiNNaker/. Accessed: 2024-05-31.

[52] Otter.ai. 2023. *Otter.ai AI-Powered Voice Transcription Service.* https://otter.ai/ Version 2023.

[53] David Piorkowski, Scott Fleming, Christopher Scaffidi, Christopher Bogart, Margaret Burnett, Bonnie John, Rachel Bellamy, and Calvin Swart. [n. d.]. Reactive information foraging: an empirical investigation of theory-based recommender systems for programmers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2012-05-05) *(CHI '12)*. Association for Computing Machinery, 1471–1480. https://doi.org/10.1145/2207676.2208608

[54] James S Plank, Bryson Gullett, Adam Z Foshie, Garrett S Rose, and Catherine D Schuman. 2022. Disclosure of a neuromorphic starter kit. *arXiv preprint arXiv:2211.04526* (2022).

[55] Thomas E. Potok, Catherine Schuman, Steven Young, Robert Patton, Federico Spedalieri, Jeremy Liu, Ke-Thia Yao, Garrett Rose, and Gangotree Chakma. 2018. A Study of Complex Deep Learning Networks on High-Performance, Neuromorphic, and Quantum Computers. *J. Emerg. Technol. Comput. Syst.* 14, 2, Article 19 (jul 2018), 21 pages. https://doi.org/10.1145/3178454

[56] Daniel Rasmussen. 2019. NengoDL: Combining deep learning and neuromorphic modelling methods. *Neuroinformatics* 17, 4 (2019), 611–628.

[57] Eldon Schoop, Forrest Huang, and Bjoern Hartmann. [n. d.]. UMLAUT: Debugging Deep Learning Programs using Program Structure and Model Behavior. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2021-05-07) *(CHI '21)*. Association for Computing Machinery, 1–16. https://doi.org/10.1145/3411764.3445538

[58] Catherine D Schuman, Shruti R Kulkarni, Maryam Parsa, J Parker Mitchell, Prasanna Date, and Bill Kay. 2022. Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science* 2, 1 (2022), 10–19.

[59] Catherine D. Schuman, Thomas E. Potok, Robert M. Patton, J. Douglas Birdwell, Mark E. Dean, Garrett S. Rose, and James S. Plank. 2017. A Survey of Neuromorphic Computing and Neural Networks in Hardware. https://doi.org/10.48550/arXiv.1705.06963 arXiv:1705.06963 [cs].

[60] Catherine D. Schuman, Thomas E. Potok, Robert M. Patton, J. Douglas Birdwell, Mark E. Dean, Garrett S. Rose, and James S. Plank. 2017. A Survey of Neuromorphic Computing and Neural Networks in Hardware. *CoRR* abs/1705.06963 (2017). arXiv:1705.06963 http://arxiv.org/abs/1705.06963

[61] Shital Shah, Roland Fernandez, and Steven Drucker. [n. d.]. A system for real-time interactive analysis of deep learning training. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (New York, NY, USA, 2019-06-18) *(EICS '19)*. Association for Computing Machinery, 1–6. https://doi.org/10.1145/3319499.3328231

[62] Felix Staudigl, Farhad Merchant, and Rainer Leupers. 2021. A survey of neuromorphic computing-in-memory: architectures, simulators, and security. *IEEE Design & Test* 39, 2 (2021), 90–99.

[63] Marcel Stimberg, Romain Brette, and Dan FM Goodman. 2019. Brian 2, an intuitive and efficient neural simulator. *elife* 8 (2019), e47314.

[64] Marcel Stimberg, Dan FM Goodman, and Thomas Nowotny. 2018. Brian2GeNN: a system for accelerating a large variety of spiking neural networks with graphics hardware. *bioRxiv* (2018), 448050.

[65] Neptune team. 2019. neptune.ai. https://neptune.ai/

[66] John Timmer. 2021. Intel launches its next-generation neuromorphic processor—so, what's that again? https://arstechnica.com/science/2021/09/understanding-neuromorphic-computing-and-why-intels-excited-about-it/

[67] Maja Videnovik, Tone Vold, Linda Kiønig, Ana Madevska Bogdanova, and Vladimir Trajkovik. 2023. Game-based learning in computer science education: a scoping literature review. *International Journal of STEM Education* 10, 1 (2023), 54.

[68] Fangxin Wang, Miao Zhang, Xiangxiang Wang, Xiaoqiang Ma, and Jiangchuan Liu. 2020. Deep Learning for Edge Computing Applications: A State-of-the-Art Survey. *IEEE Access* 8 (2020), 58322–58336. https://doi.org/10.1109/ACCESS.2020.2982411

[69] Zijie J Wang, Robert Turko, Omar Shaikh, Haekyu Park, Nilaksh Das, Fred Hohman, Minsuk Kahng, and Duen Horng Chau. 2020. CNN 101: Interactive visual learning for convolutional neural networks. In *Extended abstracts of the 2020 CHI conference on human factors in computing systems*. 1–7.

[70] Mark Weiser. 1999. The Computer for the 21st Century. *SIGMOBILE Mob. Comput. Commun. Rev.* 3, 3 (jul 1999), 3–11. https://doi.org/10.1145/329124.329126

[71] Sam (Likun) Xi, Yuan Yao, Kshitij Bhardwaj, Paul Whatmough, Gu-Yeon Wei, and David Brooks. [n. d.]. SMAUG: End-to-End Full-Stack Simulation Infrastructure for Deep Learning Workloads. 17, 4 ([n. d.]), 39:1–39:26. https://doi.org/10.1145/3424669

[72] Litao Yan, Elena L. Glassman, and Tianyi Zhang. 2021. Visualizing Examples of Deep Neural Networks at Scale. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2021-05-07) *(CHI '21)*. Association for Computing Machinery, 1–14. https://doi.org/10.1145/3411764.3445654

[73] Litao Yan, Alyssa Hwang, Zhiyuan Wu, and Andrew Head. [n. d.]. Ivie: Lightweight Anchored Explanations of Just-Generated Code. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (Honolulu HI USA, 2024-05-11). ACM, 1–15. https://doi.org/10.1145/3613904.3642239

[74] Qian Yang, Jina Suh, Nan-Chen Chen, and Gonzalo Ramos. 2018. Grounding Interactive Machine Learning Tool Design in How Non-Experts Actually Build Models. In *Proceedings of the 2018 Designing Interactive Systems Conference* (Hong Kong, China) *(DIS '18)*. Association for Computing Machinery, New York, NY, USA, 573–584. https://doi.org/10.1145/3196709.3196729

[75] Esin Yavuz, James Turner, and Thomas Nowotny. 2016. GeNN: a code generation framework for accelerated brain simulations. *Scientific reports* 6, 1 (2016), 18854.

[76] Jason Yik, Soikat Hasan Ahmed, Zergham Ahmed, Brian Anderson, Andreas G Andreou, Chiara Bartolozzi, Arindam Basu, Douwe den Blanken, Petrut Bogdan, Sonia Buckley, et al. 2023. Neurobench: Advancing neuromorphic computing through collaborative, fair and representative benchmarking. (2023).

[77] Jason Yik, Korneel Van den Berghe, Douwe den Blanken, Younes Bouhadjar, Maxime Fabre, Paul Hueber, Denis Kleyko, Noah Pacik-Nelson, Pao-Sheng Vincent Sun, Guangzhi Tang, Shenqi Wang, Biyan Zhou, Soikat Hasan Ahmed, George Vathakkattil Joseph, Benedetto Leto, Aurora Micheli, Anurag Kumar Mishra, Gregor Lenz, Tao Sun, Zergham Ahmed, Mahmoud Akl, Brian Anderson, Andreas G. Andreou, Chiara Bartolozzi, Arindam Basu, Petrut Bogdan, Sander Bohte, Sonia Buckley, Gert Cauwenberghs, Elisabetta Chicca, Federico Corradi, Guido de Croon, Andreea Danielescu, Anurag Daram, Mike Davies, Yigit Demirag, Jason Eshraghian, Tobias Fischer, Jeremy Forest, Vittorio Fra, Steve Furber, P. Michael Furlong, William Gilpin, Aditya Gilra, Hector A. Gonzalez, Giacomo Indiveri, Siddharth Joshi, Vedant Karia, Lyes Khacef, James C. Knight, Laura Kriener, Rajkumar Kubendran, Dhireesha Kudithipudi, Yao-Hong Liu, Shih-Chii Liu, Haoyuan Ma, Rajit Manohar, Josep Maria Margarit-Taulé, Christian Mayr, Konstantinos Michmizos, Dylan Muir, Emre Neftci, Thomas Nowotny, Fabrizio Ottati, Ayca Ozcelikkale, Priyadarshini Panda, Jongkil Park, Melika Payvand, Christian Pehle, Mihai A. Petrovici, Alessandro Pierro, Christoph Posch, Alpha Renner, Yulia Sandamirskaya, Clemens JS Schaefer, André van Schaik, Johannes Schemmel, Samuel Schmidgall, Catherine Schuman, Jae sun Seo, Sadique Sheik, Sumit Bam Shrestha, Manolis Sifalakis, Amos Sironi, Matthew Stewart, Kenneth Stewart, Terrence C. Stewart, Philipp Stratmann, Jonathan Timcheck, Nergis Tömen, Gianvito Urgese, Marian Verhelst, Craig M. Vineyard, Bernhard Vogginger, Amirreza Yousefzadeh, Fatima Tuz Zohora, Charlotte Frenkel, and Vijay Janapa Reddi. 2024. NeuroBench: A Framework for Benchmarking Neuromorphic Computing Algorithms and Systems. arXiv:2304.04640 [cs.AI] https://arxiv.org/abs/2304.04640

[78] Geoffrey X. Yu, Tovi Grossman, and Gennady Pekhimenko. [n. d.]. Skyline: Interactive In-Editor Computational Performance Profiling for Deep Neural Network Training. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 2020-10-20) *(UIST '20)*. Association for Computing Machinery, 126–139. https://doi.org/10.1145/3379337.3415890

[79] Tong Yu and Hong Zhu. 2020. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689* (2020).

[80] Mohamed Zahran. 2016. Heterogeneous computing: Here to stay: Hardware and software perspectives. *Queue* 14, 6 (2016), 31–42.

[81] Tianyi Zhang, Cuiyun Gao, Lei Ma, Michael Lyu, and Miryung Kim. 2019. An Empirical Study of Common Challenges in Developing Deep Learning Applications. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. 104–115. https://doi.org/10.1109/ISSRE.2019.00020

[82] Lina Zhou, Shimei Pan, Jianwu Wang, and Athanasios V. Vasilakos. 2017. Machine learning on big data: Opportunities and challenges. *Neurocomputing* 237 (2017), 350–361. https://doi.org/10.1016/j.neucom.2017.01.026

A Interview Questionnaire

A.1 Demographic Questions/Background

In this first section, we will talk about you, your background and experience.

1. What is your current role?

2. What led you to the field of neuromorphic computing? - Your background, and what degrees do you have? What did you major in?

3. How many years of programming experience do you have?

4. How many years of experience do you have as a developer?

5. What experience do you have in building ML applications? In building Deep learning applications? And in building neuromorphic computing applications?

6. Are there any other areas that you have focused on besides AI/ML?

7. What language do you use to build applications with Neuromorphic programming?

8. What tools or IDEs do you use to build applications with Neuromorphic programming?

9. In what ways, if any, are these different from those in traditional programming?

A.2 Walkthrough Building an App

For the next few questions, think about a neuromorphic computing algorithm or application you've recently built.

10.What were you trying to accomplish in building it?

11. Describe your workflow in accomplishing this.

12. What were some of the challenges you faced in building this algorithm or application?

13. What resources (e.g., websites, APIs, and documentation) did you use to answer questions?

A.3 Specific Challenges

14. What challenges do you have working with spikes as the primary signal?

- In comparison to conventional programming, what challenges does working with spikes bring, in the algorithms, data structures, patterns you create or the mental models you use?
- Since the current conventions of programming are influenced by the Von-Neumann architecture, what are the challenges when you program for a system that adopts a Neuromorphic architecture?

15. What differences are there when running code on hardware rather than in a simulator? What challenges may this bring?

16. Have you performed any optimization on your algorithms? What are the barriers you faced during such optimization?

17. When you have a challenge, what external resources or people do you use to help address these challenges?

18. How does the development time in neuromorphic computing compare with standard implementation?

19. In terms of resource availability, how does neuromorphic compare with respect to standard implementation?

20. Are there resources you'd like to see that don't yet exist?

A.4 Potential Solutions

21. What types of tools or libraries would be helpful? How might they differ from the tools or libraries that exist now? APIs?

22. If there was a simple starter application to illustrate neuromorphic computing, what would it demonstrate? E.g.: A 'Hello World' program in Neuromorphic Computing, like: Print a number, 'n', that is input by the user for n number of times.

23. What new strategies or skills have you developed to help cope with the challenges of neuromorphic computing?