INVITED PAPER

Advancing Neuromorphic Computing With Loihi: A Survey of Results and Outlook

This article provides a survey of results obtained to date with Intel's Loihi across the major algorithmic domains under study, including deep-learning approaches as well as novel approaches that aim to harness the key features of spike-based neuromorphic hardware more directly.

By Mike Davies, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A. Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R. Risbud

ABSTRACT | Deep artificial neural networks apply principles of the brain's information processing that led to breakthroughs in machine learning spanning many problem domains. Neuromorphic computing aims to take this a step further to chips more directly inspired by the form and function of biological neural circuits, so they can process new knowledge, adapt, behave, and learn in real time at low power levels. Despite several decades of research, until recently, very few published results have shown that today's neuromorphic chips can demonstrate quantitative computational value. This is now changing with the advent of Intel's Loihi, a neuromorphic research processor designed to support a broad range of spiking neural networks with sufficient scale, performance, and features to deliver competitive results compared to state-of-the-art contemporary computing architectures. This survey reviews results that are obtained to date with Loihi across the major algorithmic domains under study, including deep learning approaches and novel approaches that aim to more directly harness the key features of spike-based neuromorphic hardware. While conventional feedforward deep neural networks show modest if any benefit on Loihi, more brain-inspired networks using recurrence, precise spike-timing relationships, synaptic plasticity, stochasticity, and sparsity perform certain computation with orders of magnitude lower latency and energy compared to state-of-the-art conventional approaches. These compelling neuromorphic networks solve a diverse range of problems representative of brain-like computation, such as event-based data processing, adaptive control, constrained optimization, sparse feature regression, and graph search.

KEYWORDS | Computer architecture; neural network hardware; neuromorphics.

I. INTRODUCTION

Neuromorphic computing seeks to understand and adapt fundamental properties of neural architectures found in nature in order to discover a new model of computer architecture, one that is natively suited for classes of brain-inspired computation that challenge the von Neumann model. These properties include fully integrated memory-and-computing, fine-grain parallelism, pervasive feedback and recurrence, massive network fan-outs, low precision and stochastic computation, and continuously adaptive processes commonly associated with learning. These properties also include sparse, spike-based interactions to mediate distributed communication. Such spiking neural networks (SNNs) naturally provide energy efficiency by preferring inactive states and low-latency processing by operating in an asynchronous, event-driven manner.

The rethinking of computing that results from this pursuit intersects in unexpected ways with relevant fields, such as machine learning, deep learning, artificial intelligence, computational science, and computer architecture. As the results in this survey show, a chip like Loihi and the workloads that it supports do not fit within a well-defined box, at least not a box that is well understood today.

Manuscript received August 7, 2020; revised January 19, 2021; accepted March 7, 2021. Date of publication April 6, 2021; date of current version April 30, 2021. This work was supported by Intel Corporation. *(Corresponding author: Mike Davies.)*

The authors are with Intel Labs, Intel Corporation, Santa Clara, CA 95054 USA (e-mail: mike.davies@intel.com).

This article has supplementary downloadable material available at https://doi.org/10.1109/JPROC.2021.3067593.

Digital Object Identifier 10.1109/JPROC.2021.3067593

Some have viewed this ambiguity of scope and definition critically, pointing to a woeful lack of a clear computational model, such as the Turing machine, to guide principled algorithm discovery. Such concerns are, of course, justified, but, taking the long view, some patience is warranted. Long before the arrival of the von Neumann architecture and Turing's mathematical abstraction, engineers were devising special-purpose computing devices, dating back to the 2000-year-old astrolabe and 18th-century mechanical calculators. Those early machines, as special purpose as they were, provided real-world value and guided the way to the programmable instruction-sequenced model of the general-purpose computer architecture that thrives today.

In the history of brain-inspired computing, one finds many chips and systems dating back even before the modern era of "neuromorphic" computing, as pioneered by Carver Mead in the 1980s. The first of which, the Perceptron Mark I [1] developed in the 1950s, implemented 16 analog neurons in a mainframe-sized cabinet enclosing over four cubic meters of wires, mechanical potentiometers, relays, and other state-of-the-art electrical devices of its day. Almost all of these systems over the years were designed as exploratory demonstrations, typically with no expectations of outperforming conventional contemporary computing technology for real-world problems.

More recent large-scale efforts include the Human Brain Project systems, SpiNNaker [2], and BrainScaleS [3], which were commissioned for the specific purpose of accelerating neuroscience simulations [4]. Despite promising quantitative experiments [5], [6], these systems have struggled to demonstrate value as a practical tool for neuroscience discovery [7]. This illustrates the challenge that the neuromorphic community faces: even with fundamental architectural advantages, it is difficult for research systems to match the mature products of conventional computing that have been optimized over generations and even co-optimized with the underlying manufacturing technology.

Traditionally, much of the hardware focus in this field has gone to implementations of biological neural mechanisms in analog circuits, usually subthreshold circuits. This has led to a long list of exotic chips over the past three decades, beginning with Mead and Mahowald's silicon retina [8] through more recent chips impressively integrating up to 65 536 analog neurons and boards with one million neurons [9] and even some with synaptic plasticity [10]. These chips have generally proven extremely difficult to work with due to lack of software support, highly constrained feature sets, small scale, and often unpredictable operation. By far, the most impactful and tangible outcome of this line of research is the event-based vision sensor technology now being commercialized by at least five companies, which traces a direct lineage back to the original Caltech silicon retina research.

IBM's TrueNorth [11] represents a milestone in neuromorphic research by showing that a highly integrated digital neuromorphic chip can achieve compelling levels of energy efficiency that many previously assumed would require analog circuits. Implemented with a digital design methodology, TrueNorth integrates one million neurons in a single chip, far surpassing all prior neuromorphic chips, and is able to support meaningful neural network inference workloads at power levels as low as 70 mW. Nevertheless, beyond generating a considerable body of proof-of-concept application examples, few, if any, of TrueNorth's published results show it outperforming contemporary state-of-the-art architectures. We speculate that this is due to its slow speed exacerbated by a restrictive feature set. For example, a recent demonstration of the locally competitive algorithm (LCA) on TrueNorth [12] operates at similar power levels as Loihi for the same size of the problem but requires six to seven orders of magnitude longer to converge to a solution as a result of the contortions necessary to run LCA on that architecture.

Loihi, first published in 2018 [13], is a research processor developed with the goal of demonstrating the computational value of neuromorphic architecture as realized with today's manufacturing technology. It has been made available to a broad research community with a software toolchain enabling rigorous performance and efficiency benchmarking. Since Loihi was developed with the modest resources of a research program and is currently being used by a small (but brave) research developer community, the value demonstrated may be considered a lower bound on what this architecture is capable of delivering.

In our research with Loihi, we have encouraged a focus on the fundamentals, theory, algorithms, and rigorous benchmarking over hasty attempts to apply, demo, or commercialize the technology. Given the many degrees of freedom in the neuromorphic exploration space, the field demands a rigorous, methodical approach for reliable progress [14]. Demonstrations that appear impressive can often obscure important caveats. We instead have focused on the fundamentals, so the full scope of the technology can be understood—both strengths and weaknesses. This allows the most promising properties and pressing challenges to be prioritized appropriately.

In March 2018, Intel released Loihi for public use with the launching of its Intel Neuromorphic Research Community. This program, which now numbers over 100 research groups around the world, has led to a growing body of quantitative results that, on the whole, confirm both the value and the novelty of neuromorphic architectures and algorithms compared to von Neumann solutions. The results point to a niche for neuromorphic chips that is different and more general than the prevailing conventional view.

This article surveys these Loihi results and aims to demarcate our understanding today of where this technology may provide practical value in the near future. In addition to covering previously published work, we also include new rigorously characterized examples that we see confirming the breadth of neuromorphic computer architecture.

The rest of this survey is structured in six sections.

- Section II provides an overview of the Loihi chip, systems, and software toolchain.
- Section III assesses the value of these systems for deep learning applications that rely on backpropagation.
- Section IV describes results of running *attractor networks*, a class of algorithms that exploit the inherent dynamics of spiking neuron models to solve problems by converging to well-defined fixed points in phase space.
- Section V describes results from more exotic *nongradient-based* algorithms that leverage time- and event-based computations to solve search, planning, and optimization problems.
- Section VI surveys the use of the algorithmic methodologies from Sections II–V for enabling specific applications, some of which approach real-world practical relevance.
- Section VII looks to the opportunities that emerge from these Loihi results and what challenges remain.

The Supplementary Material provide more details about the examples, results, and methodologies covered in this survey.

II. LOIHI, SYSTEMS, AND SOFTWARE

Here, we provide a brief overview of the Loihi chip, systems, and software stack as a foundation for the results that follow. Interested readers are encouraged to refer to prior publications [13], [15], [16] and Intel's online resources[1] for further details.

A. Loihi Chip

Loihi implements 131 072 leaky-integrate-and-fire neurons using a digital, discrete-time computational model partitioned over 128 cores that are integrated into a spatial, asynchronous mesh. Each core contains 128 kB of synaptic state, and another 20 kB of routing tables that can be flexibly allocated over its 1024 neurons, with network compression and weight sharing mechanisms to support the largest and most complex networks possible. All communication between neurons occurs over spike events, 32-bit messages containing destination addressing, and, sometimes, source addressing and graded-value payloads that the network-on-chip routes between cores. Each core is responsible for sending generated spikes to all downstream cores containing fan-out neurons and replicating all ingress spikes to its associated fan-out neurons, based on configured routing information. Weights and delays associated with each synaptic connection control how the replicated spikes are applied to the attached postsynaptic neurons.

Numerous novel features distinguish Loihi from other neuromorphic chips. Its synaptic memory is highly configurable, supporting not just compression and weight sharing but variable weight precision (from 1- to signed 9-b values), delays of up to 63 timesteps that are applied uniquely to source–destination neuron pairs, and synaptic scratch variables called *tags*, inspired from biological models of reinforcement learning, that serve as auxiliary dynamic state variables associated with a synapse.

Plasticity rules may be specified by microcode and assigned to synapses such that their state variables can, if desired, programmatically evolve over time as a result of presynaptic and postsynaptic spike activities that are locally maintained and accessible to the synapse. For example, Loihi supports the classic *Bi and Poo* Spike Timing Dependent Plasticity rule, but it also supports a wide range of other rules, such as rate-based Hebbian rules, reward-modulated rules, and rules that mix activities filtered on different timescales. Loihi's plasticity rules have found a use for a variety of adaptation, learning, and other applications, sometimes in surprising ways. For example, our graph search algorithm described in Section V-B uses weight plasticity in conjunction with synaptic delays to identify the shortest path in a given weighted graph.

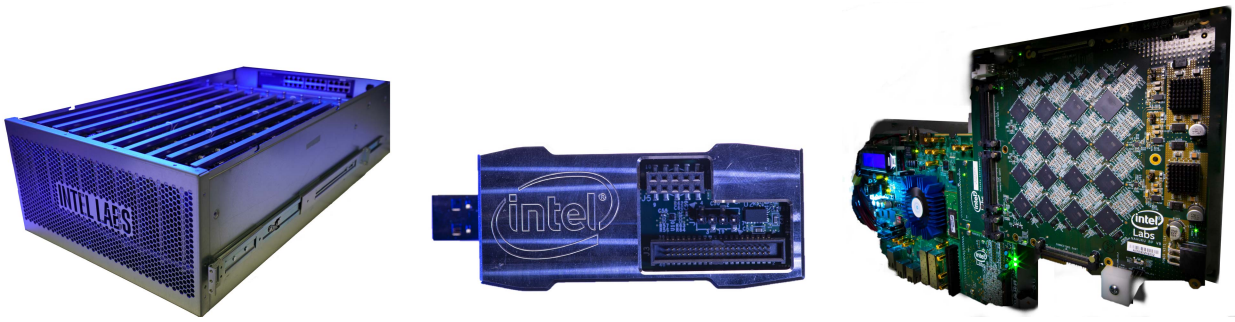
Within a core, neurons may be distributed over multiple *compartments* or dynamic state variables, each with uniquely configured filtering dynamics. Compartments communicate integer-value (graded) state variables over tree topologies, analogous to a dendritic tree, and optionally generate spikes for communicating significant events to other neurons. An example use of this multicompartment feature is described in Section V-C, where they are used to implement an online constraint satisfaction solver.

Other neuroinspired Loihi features include graded reward spikes that modulate learning rules, axon and refractory delays, pseudorandom noise that may be applied to various neuron state variables, and a threshold adaptation mechanism.

Loihi's neuromorphic mesh and cores are built with asynchronous circuits and, at the transistor level, communicate event-driven tokens of information between logic stages. This allows spike messages and iterative processes within each core to proceed as fast or slow as the computation and pipeline activities allow without ever waiting for clock edges or needlessly expending clock power during periods of inactivity. Loihi's asynchronous handshaking extends to four off-chip interfaces that scale the 2-D on-chip mesh into a similar second-level interchip mesh. At their source, spikes destined for other chips are encapsulated with a 32-bit header that specifies the necessary extra chip addressing.

In natural brains, various feedback processes introduce synchronization and coherent information processing over different neurons and brain regions. Loihi implements a similar but comparatively brute force emergent synchronization mechanism using periodic wavefronts of *barrier* messages that may be viewed as a special category of spikes. This barrier synchronization process allows all chips and cores in a multichip mesh to operate independently but, through barrier-mediated handshaking, still

[1] https://www.intel.com/content/www/us/en/research/neuromorphic-community.html



(a) (b) (c)

Fig. 1. *Loihi systems. (a) Pohoiki Springs large-scale system with 768 Loihi chips. (b) Kapoho Bay USB form factor with two Loihi chips, plus sensor, and event-based camera AER interfaces. (c) Nahuku 32-chip expansion board interfaced to the Intel Arria 10 FPGA development system. Images copyright 2020 Intel, used with permission.*

stay sufficiently synchronized in order to respect the discrete-time computational model and ensure deterministic operation. Depending on application needs, timesteps may be throttled to a real-time scale (one millisecond per timestep is common), or the mesh may operate unthrottled, proceeding as fast as the communication patterns in the workload allow. The latter results in each timestep consuming varying amounts of real time.

An important, perhaps surprising, ingredient in Loihi is von Neumann processing. Each Loihi chip instantiates three microcontroller-class x86 processors at the periphery of the mesh that offloads management tasks from the primary host CPU that is too frequent to run efficiently off-chip, as well as SNN algorithmic processes that are too infrequent to justify implementing in the neuromorphic cores. The x86 cores are often used for data format conversion, bridging between the dense, synchronous encodings of conventional computing, and the spike- and event-based encodings of the neuromorphic domain.

In space, time, and connectivity, Loihi's architecture is optimized for sparse and nonbatched computation. The Loihi cores have narrow datapaths and memory word sizes. Memory access is always local and hypergranular, so data-dependent control flow is fast and efficient. Outside the core, all spike messages carry events related to the activity of a single-source neuron, putting no pressure on the architecture or algorithms to maintain activity over blocks of neurons with shared connectivity. These properties place Loihi in a diametrically opposite architectural regime compared to state-of-the-art von Neumann processors and deep learning accelerators, whose wide datapaths, deep pipelines, and high memory access latencies demand dense, deep, and predictably active networks in order to achieve high performance and efficiency.

For more details on the Loihi chip architecture and implementation, we refer the readers to Davies *et al.* [13].

B. Systems

Interfacing Loihi to conventional computer systems requires bridging between the asynchronous, event-based communication protocols of the neuromorphic domain and the standard synchronous protocols of a host CPU and peripherals. Any future commercialized form of the architecture would integrate standard interfaces on-chip, but, for Loihi, such straightforward engineering work was outside the scope of what a small research team could implement. As a consequence, the conversion must happen off-chip in a field-programmable gate array (FPGA) device, which introduces power and latency overhead whenever a CPU transfers data to and from Loihi.

Three types of Loihi systems are available for use by researchers, either through remote cloud access or on-site in their labs:

- *Kapoho Bay [see Fig. 1(b)]:* A USB form factor device. In addition to USB connectivity, this device includes a second FPGA that exposes the Loihi mesh interface in address event representation (AER) [17] form compatible with the IniVation DAVIS240C [18] event-based camera and offers extensibility for other hardware sensors and actuators.
- *Nahuku [see Fig. 1(c)]:* A 32-chip Loihi board provisioned with power supplies, power measurement circuitry, and a standard FMC connector that allows it to be interfaced to an Arria 10 FPGA development board. The Arria 10's embedded ARM CPU, in coordination with a more distant Intel CPU connected over Ethernet, handles data I/O and programming of the Loihi mesh. The Arria 10 also provides interfaces to the DAVIS 240C camera and other standard peripheral devices.
- *Pohoiki Springs [Fig. 1(a)]:* A rack-mounted chassis enclosing 768 Loihi chips, FPGA interface boards, and an integrated IA host CPU [16]. Pohoiki Springs implements 100 million neurons in five standard server rack units, corresponding to over 100 million times less space per neuron than the Perceptron Mark I. It typically operates at under 300 W, which is less power than what most single-CPU datacenter servers require.

C. Software

The Loihi system architecture is a spatially distributed, heterogeneous collection of computing elements that calls

for a unique software framework to make the overall system functional and performant. No existing mainstream framework provides suitable abstractions that span neuromorphic cores, embedded x86 processors, FPGA I/O interfaces, and tiers of host CPUs. The unifying principle of the system is event-based asynchronous communication. Across higher levels, communication occurs with message passing over statically allocated channels using standard interfaces, such as Ethernet and USB, with messages buffered in the memories of host von Neumann processors. At the other extreme, within the neuromorphic mesh, the granularity of messages is much smaller with nanosecond-timescale messages communicating single-bit spike events between neurons routed over hardware channels with no buffering.

We refer to the set of software tools for this system as *NxSDK*. The framework provides APIs, compilers, and debugging tools for programming Loihi, as one would expect from any SDK, but it also includes other necessary ingredients, such as a runtime for the lower layers and interfacing to a variety of third-party frameworks at both the development and runtime levels.

Coding for this system involves a mixture of declarative programming to specify the structure of the networks and imperative programming of conventional code that runs on the von Neumann processors. NxSDK defines a special category of processes in the latter category called *snips*, or sequential neural interfacing processes, that communicate over channels and are relocatable over the embedded x86 processors in the Loihi mesh. Snips are responsible for interacting with neurons, often handling real-time data format conversion, sequencing different phases of operation, such as learning and inference, and configuring neuron parameters as needed.

NxSDK allows users to specify a network at various levels of abstraction, from the lowest level where neuron parameters are individually controlled, to an intermediate level where neurons or groups of neurons are defined by their desired behavior, to the highest level where the network itself is abstracted away inside a prepackaged parameterized module designed to perform a specific task.

NxSDK also acts as a backend for several third-party frameworks that allow users to specify networks in a language that they are more familiar with. The neural engineering framework (NEF) [19] and other related capabilities are supported through the Nengo toolchain [20] from Applied Brain Research. PyTorch and Tensorflow models are supported through the SNN Conversion Toolbox [21], and directly training deep SNNs is supported through SLAYER [22].

Beyond interfacing with the spiking neurons running on Loihi, snips on the host CPU provide real-time interfaces to a variety of robotic and simulation frameworks. To date, interfaces have been implemented for the Robot Operating System (ROS), Yet Another Robot Platform (YARP), and simulators, such as Mujoco, Gazebo, and the Neurorobotics Platform [23].

III. DEEP LEARNING FOR SPIKING NEURAL NETWORKS

Deep learning offers a natural starting point for SNN research. The basic deep learning paradigm of applying the error backpropagation algorithm to differentiable artificial neural networks (ANNs) has proven to be a powerful tool for optimizing these high-dimensional, nonlinear models with precollected data sets to solve a wide range of problems. Given the success of deep learning for ANNs, one may reasonably hope that the same approach would also yield successes for SNNs.

However, before proceeding, we must first resolve a common misconception that arises in this area. Many incorrectly assume that SNN chips, such as Loihi, are designed specifically to accelerate standard deep learning models, such as MobileNets and ResNets, with the aim being to outperform GPUs and ASICs in energy efficiency and speed on these workloads. Based on this incorrect assumption, the focus turns to compare the energy and latency of the multiply-and-accumulate (MAC) operation required by ANNs to the analogous "synaptic operations" in SNNs. This view completely ignores the time and energy cost of implementing the internal dynamics of spiking neurons, which add important computational capabilities to an SNN, but is not used by ANNs.

Even if we just focus on comparing MACs, the argument goes that the synaptic operation is a simpler accumulation operation than the MAC and should, therefore, reduce energy. While this is true in principle, the energy for a synaptic operation on Loihi is greater than a MAC in a typical custom ANN accelerator. This comes from the overhead of supporting sparse network architectures, a need that goes hand-in-hand with supporting sparse activation in time- and event-driven computations. In contrast, today's state-of-the-art GPUs and ANN accelerators achieve extremely low MAC energies with highly vectorized datapaths and communication channels designed for streaming dense batches of data. Furthermore, approximating a single MAC in an SNN typically requires many synaptic operations spread out over time (with rate coding), resulting in even higher energy and latency.

In short, it would be naïve to ignore ANN literature as we search for effective SNN architectures, but it would be equally naïve to expect SNNs to outperform ANN accelerators on the very task that they have been optimized for. Deep learning offers a fine starting point in a journey of SNN algorithm discovery, but it represents only one niche of the algorithmic universe available to neuromorphic hardware.

Deep learning-inspired SNN algorithms broadly fall into two main categories (Fig. 2): *online* approaches and *offline* approaches. *Online* approaches first deploy an SNN to neuromorphic hardware and then use on-chip plasticity features to approximate the backpropagation algorithm

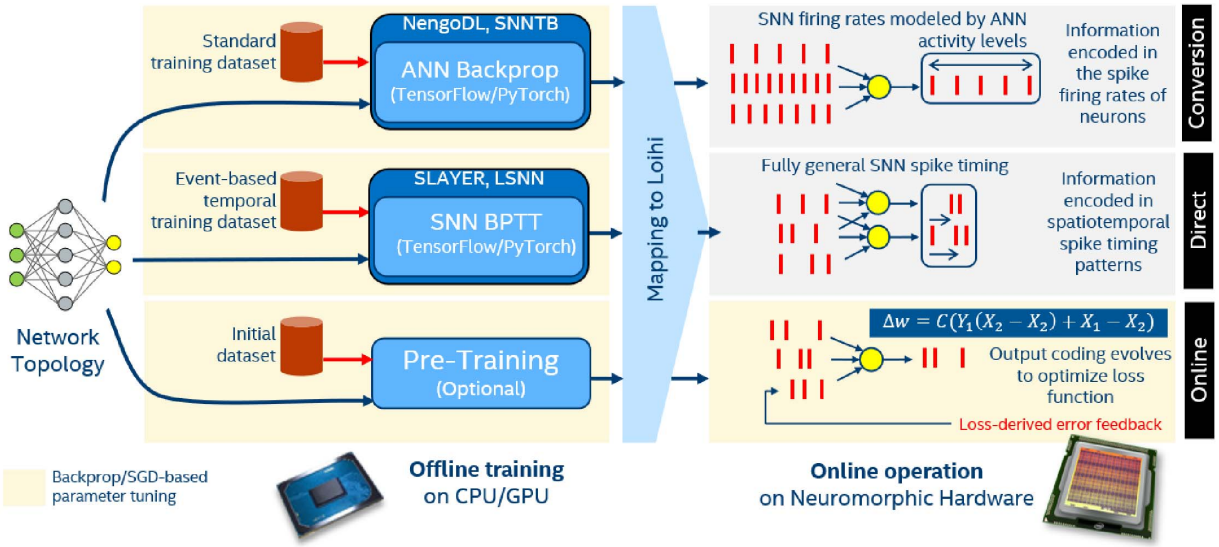


Fig. 2. ***Deep learning methodologies for SNNs.***

and training process itself, evolving the network parameters in real-time and in place as data arrives. *Offline* approaches use a prerecorded data set to train a model on a CPU or GPU and later deploy the trained model to neuromorphic hardware. Offline approaches include *conversion* approaches that convert a given ANN into an approximately equivalent SNN and *direct* approaches that build an exact ANN model using the SNN parameters and then directly learn the SNN parameters by applying backpropagation to the ANN model.

The Loihi research community has investigated a variety of deep SNN topologies and workloads using offline-conversion, offline-direct, and online training approaches. In the following review, we focus on examples that have been rigorously benchmarked in accuracy, energy, and time to solution (delay) against comparable or equivalent ANN solutions running on conventional architectures.

In all examples, the accuracy of the trained SNNs matches the accuracy of the reference implementations. However, energy and delay comparisons vary widely depending on the training approach, task complexity, and implementation details. Fig. 3 summarizes energy and time to solution ratios for eight tasks spanning from single-core to multichip workloads. The diagonal line in Fig. 3 marks energy–delay-product (EDP) parity above or below which Loihi either outperforms or underperforms the given reference architecture, respectively. Some tasks are compared on multiple reference architectures that are distinguished by different markers.

Most comparisons against Loihi use a batch size of 1 (solid markers), which is beneficial for real-time tasks that demand a low-latency response to new data. Increasing the batch size requires waiting for more data to fill up the batch before processing begins, thus increasing latency. However, a lower response latency does not necessarily equate to higher throughput. Batched and pipelined architectures might have longer latencies, but they achieve very high throughput by processing many samples at once, whereas Loihi only processes one at a time. The EDP of

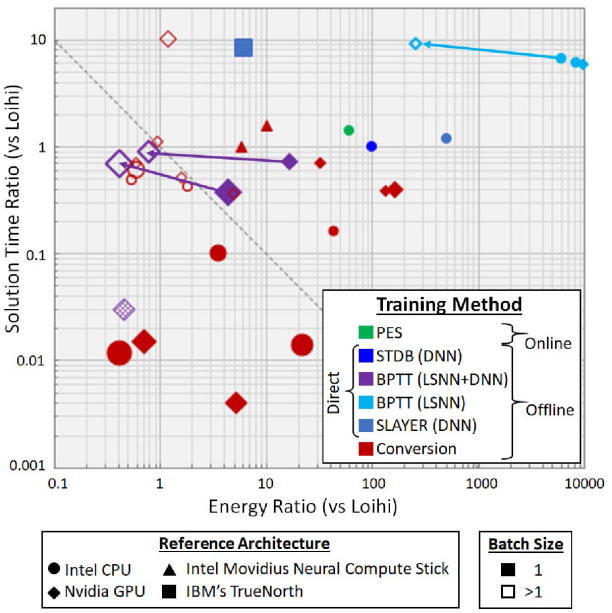


Fig. 3. ***Ratio of energy and delay between Loihi and reference architectures for all deep SNNs benchmarked to date. Loihi achieves similar accuracy to the reference architecture in all cases. Marker shape indicates the reference architecture, marker size indicates the number of Loihi cores required (log scale from 1 to 2320), and marker color indicates the training method. Loihi always uses batch size 1, but reference architectures can use a batch size of 1 (solid markers) or many (hollow markers). Arrows indicate the reference architecture's improvement when batch sizes greater than 1 are used. See Section S.I in the Supplementary Material for full workload details. Points to the upper right of the diagonal line are examples that run with superior EDP on Loihi; points to the bottom left run with worse EDP on Loihi.***

both CPU and GPU architectures improves significantly when batching is used (hollow markers).

The remainder of this section describes the general trends and lessons that emerge from these examples. Section S.I in the Supplementary Material provides more details about the methodologies and individual workloads.

A. Deep SNN Conversion

Many frameworks achieve near-lossless conversion of ANNs to SNNs on traditional image classification tasks, such as CIFAR or ImageNet [21], [24]–[26]. ANN-to-SNN conversion involves mapping the trained ANN parameters into corresponding SNN parameters. The conversion methodology ensures that the resulting SNN behavior matches the ANN's behavior according to some spike-based information encoding model. As with conventional low-power DNN accelerators, mapping parameters to Loihi requires quantization, which can introduce errors.

Conversion frameworks typically represent a continuous-valued ANN activation as a spike rate in an SNN. While an ANN may process a single static input vector, such as an image frame, through a series of dense operations, the SNN performs a series of sparse computations over multiple time steps or iterations. This temporally "unrolled" computation can be a useful feature of SNNs because it supports a dynamic tradeoff between classification accuracy and inference latency (see Fig. S1 in the Supplementary Material). However, when using a spike rate to represent a neural activation, additional bits of precision cost exponentially more encoding time.

Two end-to-end DNN conversion frameworks are currently available for Loihi: NengoDL [24] and the SNN Conversion Toolbox [21]. NengoDL invokes TensorFlow to train an ANN offline, which is then converted to a rate-coded SNN and optionally composed with other SNN modules trained using Nengo's NEF. The SNN Conversion Toolbox ingests ANNs from frameworks, such as TensorFlow, PyTorch, or Cafe, converts them to Loihi-compatible deep SNNs using a custom Loihi backend, and maps the resulting SNN to Loihi using NxTF [27].

Fig. 3 compares the latency and energy efficiency of converted deep SNNs running on Loihi against the original ANNs running on conventional hardware architectures. The tasks include audio keyword spotting [28], CIFAR image classification with MobileNet convolutional networks, generation of embeddings for similarity search [29], and segmentation of ISBI cell images using a modified U-Net architecture [30].

Loihi is substantially more energy efficient on almost all of these workloads (up to 100×). For small workloads that use only a few Loihi cores (red, above parity line), Loihi's delay is on par with the reference architectures, even for architectures that are specifically designed for batch size 1 inference, such as the Movidius Neural Compute Stick (triangles). In contrast, large-scale DNN workloads, especially those spanning multiple chips, take significantly longer on Loihi (red, below parity line) than on the reference architectures.

Loihi's poor latency scaling with increasing workload size results from two factors. First, higher layer counts demand an increasing number of time steps in order to achieve maximum accuracy. This is a well-understood and fundamental property of rate-coded feedforward networks [21], [31], [32]. Second, the need to distribute larger networks across multiple Loihi chips leads to congestion in the links between the chips and often a dramatic increase in execution time per time step. Converted deep networks, with dense cross-sectional connectivity and high aggregate spike rates, stress Loihi's off-chip mesh links, which have roughly 30× lower bandwidth than its on-chip links.

B. Direct Deep SNN Training

Direct SNN training uses backpropagation to directly optimize the parameters of an SNN. This relies on formulating the SNN as an equivalent ANN with binary input, a discontinuous spike generation function as the ANN's nonlinearity, and self-recurrent connections to model the spiking neuron subthreshold state dynamics. This formulation of SNN error backpropagation has been successfully demonstrated in recent years [22], [33]–[36], and several implementations have been applied to Loihi networks.

Direct training approaches lead to emergent spike-timing codes that better optimize latency and energy efficiency and are, therefore, of particular interest when information is encoded in the relative timing between input spikes, such as in data produced by event-based neuromorphic sensors. More generally, exploiting the temporal domain to encode information leads to more efficient information encoding than rate coding and accordingly fewer spikes, lower latency, and lower energy per computation. In fact, latencies and spike counts can be explicitly prioritized by the loss function.

While highly effective for small networks, directly training large networks proves challenging. Self-recurrence turns training into a temporal credit assignment problem and, therefore, requires backpropagation through time (BPTT), treating the SNN as a recurrent ANN. Each time step of the SNN represents one iteration of the recurrent ANN, which leads to an enormous increase in the time and memory footprint of training compared to training a feedforward ANN of the same size. Furthermore, backpropagation requires calculating the derivative of the discontinuous spike function, which is undefined. Instead, a surrogate function is used in the derivative's place, an approximation that introduces errors that accumulate with increasing network size.

Directly trained SNNs benchmarked on Loihi to date have used the Spike Layer Error Reassignment (SLAYER) algorithm [22], spatiotemporal backpropagation (STDB) [37], and a combination of BPTT with deep rewiring proposed in [36], which trains recurrent

long short-term SNNs (LSNNs) with capabilities similar to conventional LSTMs.

SLAYER was used to train a network for DVS Gesture recognition, a task first proposed in [33] as an SNN on TrueNorth. Loihi's greater flexibility allows for a smaller network than TrueNorth, which combines with a faster operation to result in a 50× lower EDP. SLAYER with Loihi has also been used for tactile digit recognition by See *et al.* [38] and for sensor fusion because of the ease of combining modalities in the spike domain. Ceolini *et al.* [39] combined EMG and vision data in a gesture classification task, and Taunyazov *et al.* [40] combined vision and tactile data in a grasping task.

STDB was used by Tang *et al.* [41] with Loihi in a robot navigation task, in which Loihi is comparable to an edge GPU in speed but lower in power, resulting in an 83× lower EDP.

We first applied LSNNs on Loihi to solve the Sequential MNIST problem, achieving 6×10^4 lower EDP than a GPU for batch size 1 and 37× compared to larger batch sizes for which GPUs are better suited. Recently, much larger collections of LSNNs interconnected with feedforward subnetworks, all trained with BPTT, have solved relational reasoning problems from the bAbI question-answering data set [42]. Running on Loihi, interchip congestion and rate coding inefficiencies from the network's feedforward components degrade its performance compared to a standalone LSNN, but the example still manages to outperform similarly structured LSTM-based solutions of the problem [43] running on a GPU. This example, which consumes up to 2320 Loihi cores, is the largest deep learning network to date showing gains compared to conventional architectures.

Results for these experiments are shown in tones of blue and purple in Fig. 3 above the EDP parity line. These workloads vary in scale from 1 to 2320 Loihi cores but always outperform the reference architecture by orders of magnitude. This outperformance illustrates the benefit of direct training to produce sparsely active and highly efficient SNNs.

C. Online Approximations of Backpropagation

Sections III-A and III-B describe offline training methods, but it is also desirable to learn online from streaming data. While backpropagation is an effective algorithm for training DNNs, it is expensive to implement in terms of time, computation, and memory, especially for BPTT [44]. A particular contributor to this expense is the locking problem [45] that originates from the dependence of a layer's update on propagating its output forward and the resulting error backward that makes native BPTT ill-suited for incremental online learning.

To harness backpropagation for neuromorphic hardware, specific simplifications of the algorithm have been proposed, which narrow its scope and approximate its gradient descent behavior, such as random feedback connections [46], synthetic gradients [45], eligibility propagation as a way to circumvent locking [47], skip connections in the backward path to avoid backpropagating data through multiple layers [48], or only adapting pretrained output layers during online learning instead of training an entire DNN from scratch.

Several of these approaches are under development for Intel's neuromorphic architecture. As the first step, instances of the delta rule[2] $\Delta w \propto x_{\text{pre}} \cdot \sigma'_{\text{post}} \cdot \delta y_{\text{post}}$ for single-layer online learning have been demonstrated on Loihi, including Surrogate Online Error Learning (SOEL) [49] and the Prescribed Error Sensitivity (PES) rule [50]. SOEL was used to train the last layer of a pretrained DVS gesture recognition network online, allowing Loihi to learn new gestures in real time. The PES learning rule was used to endow an SNN robotic arm controller with the ability to adapt to changes in real-time, such as when the arm picks up a heavy object. The approach is currently being evaluated for use as an assistive robotic arm controller for wheelchair users. Using the PES rule, Loihi outperforms an alternative CPU implementation by around 100× in EDP (see Fig. 3).

IV. ATTRACTOR NETWORKS

Unlike the standard artificial neuron, spiking neurons have temporal behavior. Therefore, networks of spiking neurons become high-dimensional, highly nonlinear dynamical systems. One defining characteristic of brains is that the computation that they perform is the result of the collective interactions between their neurons, an emergent phenomenon, such as eddies in a stream. This is fundamentally different from conventional models of computing, including ANNs, that are defined by precise, fully comprehensive, and usually sequentially formulated specifications of their behavior. Spiking neurons in the brain have no such precise model. Through feedback, adaptation, and interactions with the environment, neurons evolve to collectively behave in some desired manner despite uncertainty or nondeterminism in the precise behavior of each individual neural unit.

In deep learning, the training process is a dynamic system exhibiting these characteristics, but the execution of a trained ANN is not. With SNNs, a much broader range of computation under study, beyond just supervised learning, depends on collective dynamics.

Attractor dynamics are the simplest form of collective dynamics that lead to useful, nontrivial computation. One rigorous strategy for developing an attractor-based SNN algorithm is to prove that the network satisfies a convergence guarantee, known as a Lyapunov condition. A Lyapunov condition does not precisely describe the network's dynamics, but its existence implies that the network will, eventually, converge to a particular well-defined equilibrium state. Sometimes, the network's equilibrium states can be mathematically characterized in closed form, as is

[2] x_{pre} refers to the presynaptic activation, σ'_{post} is the postsynaptic pseudoderivative of the activation function, and δy_{post} is the postsynaptic error.

the case for the LCA described in the following, but, even when not, the network may behave in an intuitively understandable manner that allows for complex attractor-based behavior to be engineered, as in the case of dynamic neural fields (DNFs) described in Section IV-B.

In short, attractor networks are a class of SNN algorithms that solve problems or provide useful behavior with their emergent dynamics based on the equilibrium states they converge to given static inputs. This class provides a fruitful next step in a journey of SNN algorithm discovery beyond the deep learning paradigm.

A. Locally Competitive Algorithm

The simplest attractor networks, for example, Hopfield networks, have a number of neurons that are connected in an all-to-all fashion with a symmetric weight matrix. The dynamics of all such networks satisfy a Lyapunov condition and will converge to a fixed point that corresponds to a minimum value of an energy function. Conventionally, these networks are defined with analog-valued rate neurons that make them mathematically tractable. For neuromorphic applications, it is often possible to rigorously link the rate neuron dynamics with the corresponding dynamics of an equivalent SNN, given a suitable choice of leaky-integrate-and-fire neuron model [51].

One simple attractor network that performs the useful nontrivial computation in its dynamics is the LCA [52]. In an LCA network, an input signal is projected to a set of feature neurons that are mutually inhibited with recurrent connections. The balance between feedforward input and recurrent inhibition induces competition in the network, and over time, the system converges to a sparsely active set of features that best explain the input. If the parameters of the network are configured according to the LCA algorithm, then the network's equilibrium state will exactly correspond to the solution of the well-known LASSO regression problem. LASSO long predates LCA and today finds wide use in statistics as a technique for reducing overfitting and identifying sparse feature sets.

Previous results [13], [53] demonstrated the efficiency of neuromorphic architectures, such as Loihi, for solving LASSO problems with LCA, especially the convolutional form of the problem.[3] Convolutional LASSO, typically applied to sparse coding of images, applies a translation-invariant dictionary over patches or windows of the input. Sharing dictionary weights between patches reduces the memory required to store weights. Unlike traditional feedforward convolutional neural networks, convolutional LCA with overlapping patches recurrently connects all feature neurons associated with all patches of an image. This creates a challenging problem for conventional gradient descent-based LASSO solvers using matrix data structures since they must process the entire feature and dictionary matrices sequentially to convergence.

[3]See Section S.II.A in the Supplementary Material for LCA network architecture.

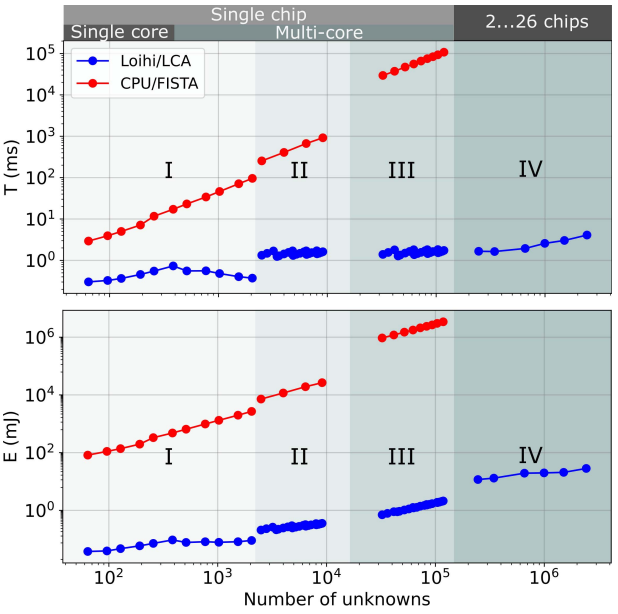


Fig. 4. ***Time-to-solution (top) and dynamic energy consumption (bottom) for two ways of generating a sparse code for an input image by solving a LASSO optimization problem: LCA implemented on Loihi (blue) and FISTA implemented on x86 CPU (red). Region I corresponds to small LASSO problems, for which a nonconvolutional variant of LCA suffices (16 × 16 pixel input image and dictionaries consisting of 32–1024 elements). On the other hand, convolutional LCA is used to solve the LASSO problems in regions II–IV. The input sizes for regions II and III are 24 × 24 and 76 × 76 pixels, respectively, with dictionaries of 50–180 elements. In region IV, input sizes are in the range of 130 × 130–300 × 300 pixels with dictionaries containing 120–250 elements. In all convolutional regions (II–IV), a patch and a stride size of 8 × 8 and 4 pixels are used, respectively. CPU implementation of FISTA is nonconvolutional. The problem sizes analyzed in region IV span multiple Loihi chips and are too large to benchmark on a CPU. We refer to LCA as task 9 in Section VIII.***

NxSDK provides a compiler for convolutional LCA networks that exploit weight sharing to make efficient use of on-chip memory and support networks spanning millions of feature neurons. Fig. 4 compares LCA on Loihi[4] to a CPU[4] running FISTA [54], the leading conventional algorithm. Both algorithms generate sparse codes of input images[5] by solving the same LASSO problem to the same quality of the solution, as measured by the LASSO objective function. Loihi LCA objective values typically saturate at ∼1% of the optimal value, which sets the convergence threshold of the evaluations. This is approximate but sufficient for many applications.[6]

[4]Loihi: Wolf Mountain board with NxSDK v0.75; CPU: Intel Core i7-4790 3.6-GHz w/ 32-GB RAM. BIOS: AMI F5. OS: Ubuntu 16.04 with HyperThreading disabled, running SPAMS solver for FISTA http://spams-devel.gforge.inria.fr/

[5]Randomly chosen images from the Oxford-IIIT Pet data set [55].

[6]Given sufficiently long run times, FISTA can solve LASSO to much lower objective values than LCA on Loihi. We also evaluated least angle regression (LARS) [56] and found that it performs worse than FISTA for our problem's sparsity levels and approximate convergence thresholds.

Regime I in Fig. 4 corresponds to small problem sizes that are solved using nonconvolutional LCA on Loihi. While the time and energy to solution increase exponentially for FISTA on the CPU, the same metrics increase only moderately at first on Loihi until a problem size of around 500 unknowns, for which the LCA network fits on a single neuron core. Beyond this point, the time to solution actually declines slightly due to increasing multicore parallelism. As a result, LCA on Loihi outperforms FISTA on CPU by one to two orders of magnitude in this regime.

Continued scaling is shown in regimes II–IV. The problem sizes considered in regimes II and III fit on a single Loihi chip but differ by an order of magnitude in the number of cores used to solve the problem. At the end of regime III, when compared against the CPU running FISTA, we observe up to five orders of magnitude advantage in time-to-solution and six orders of magnitude advantage in energy consumption for LCA on Loihi for the largest problem sizes of $\sim 10^5$ unknowns, as shown in Fig. 4.

FISTA does not exploit the convolutional structure of these problems[7] and, therefore, has a bloated memory footprint compared to Loihi. However, profiling revealed that FISTA's performance would be relatively unaffected by dictionary feature sharing. Its performance is limited by internal control flow, not DRAM access latency or bandwidth, over any of the tested regimes. DRAM-related power is excluded from our calculation of the CPU's dynamic energy in Fig. 4.

Regime IV corresponds to LCA networks distributed over two to 26 chips, with input sizes approaching those typically seen in real-world applications. Fig. 4 shows a slowdown in convergence as the growing LCA networks are scaled over an increasing number of chips. The slowdown is almost entirely due to spike congestion on Loihi's chip-to-chip interfaces. Nevertheless, scaling in regime IV remains superior to the CPU's general trend extrapolated from regimes I–III. Spike congestion is not unique to large LCA workloads; it is also seen in deep spiking networks.

LCA provides one of the best illustrations of a fine-grained parallel algorithm with sparse activation leveraging the matching properties of neuromorphic architecture to achieve order of magnitude gains. With highly vectorized datapaths suffering large penalties for bit-level data-dependent branching, conventional architectures are unable to efficiently exploit the algorithm's sparse activity and fine-grain parallelism. We refer the readers to [57] for details of this example.

Looking ahead to applications of LCA, sparse coding attracts interest for real-time feature extraction as part of visual processing pipelines, so much as to motivate research into fast feedforward approximations of LASSO [58]. One study showed that neuromorphic sparse coding can provide some protection from adversarial images, with Loihi's LCA implementation giving results on par with CPU-based full precision algorithms [59].

[7]See Section S.II.B in the Supplementary Material.

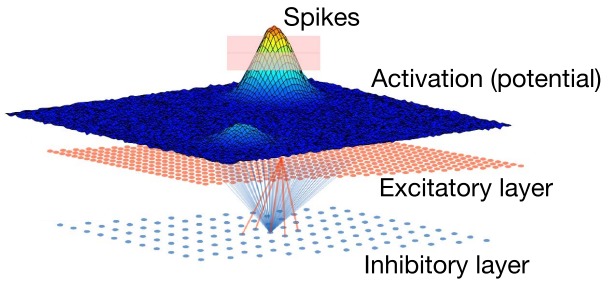


Fig. 5. *Connectivity motive of a DNF and the activation profile (membrane potential) of neurons in the excitatory layer. The spiking activity only forms over a small strongly activated region; it can be sustained if the external input ceases.*

This suggests a natural deployment of LCA as the first processing layer of such a neuromorphic pipeline. This also raises the need for online, unsupervised dictionary learning to form efficient sparse representations of sensory input. To date, numerous approaches for dictionary learning with SNNs have been proposed [60]–[62] with work underway to map these and further generalizations to Loihi.

B. Dynamic Neural Fields

DNFs provide a modular algorithmic framework for implementing states, relations, behavior, and memory using neural attractor networks. In the DNF framework, attractor states correspond to behavioral attractors, such as the location of an object or the velocity of ego-motion [63]. A particular DNF network typically implements a single behavioral variable with all-to-all connectivity, where excitation from input stimulation is balanced by inhibition from active states, as in a winner-take-all (WTA) network (see Fig. 5). The most strongly stimulated state or states manifest as persistent activation that discourages the expression of other less stimulated states, a working-memory-like construct. In addition, local excitatory connections between neurons with similar receptive fields stabilize the manifold structure of the neural field [64]. Neuroscience studies have found DNFs to successfully model many cognitive brain processes that require working memory [63].

The DNF computational framework has been proposed as an intuitive programming abstraction for neuromorphic hardware to facilitate computing in neural state machines [64], [65]. Stable states help to cope with intrinsic variability of analog neuromorphic circuits and sensory noise [66]. Each DNF unit is defined by a small number of parameters that determine its dynamical behavior, e.g., single- or multipeak solutions, persistent, or input-driven activation.

Although DNFs have been used as a programming framework for autonomous systems and cognitive robots, such as the C++ toolbox Cedar [67], their high computational cost has hindered scaling to useful real-world

tasks. Similar to LCA, sparse activity and pervasively recurrent connections prevent them from running efficiently on conventional CPU, GPU, or DNN accelerator architectures. They are, however, very well suited for Loihi's central features, such as fine-grained parallelism, sparse event-based processing, and effective weight sharing.

A two-layer 2-D DNF network has been implemented on Loihi to track a moving object viewed by a DVS event-based camera [68]. The first layer in this network filters out sensor noise and forms multiple activity "bumps" over locations of moving objects. In the second layer, the DNF tracks the selected object, inhibiting other distractor objects, even if the tracked object stops or the distractor events strengthen or briefly occlude the tracked object. The network on Loihi reliably tracks objects with a precision of 3.5 pixels when processing 240×180 input on a 64×64 neural grid in real time. This simple module can be used to track visual features on Loihi and may serve as a preprocessing step for visual odometry and SLAM (see Section VI-C), for setting an attentional focus on objects in complex scenes, or to visually track navigation targets [69].

V. COMPUTING WITH TIME

The results from applying backpropagation directly to SNNs (see Section III-B) hint at the gains in efficiency and speed that may come from optimizing SNNs to encode and process information using spatiotemporal spike patterns. While backpropagation and other gradient-based optimization techniques can exploit fine-scale timing when given a suitable differentiable network architecture, to more fully explore the very large space of networks that compute with precise spike timing relationships, especially those involving dynamic state, delay, plasticity, and stochasticity, we must cast a wider net.

A number of handcrafted SNN algorithms have been proposed in recent years to solve well-defined computational problems using spike-based temporal information processing. When implemented on neuromorphic architectures, these algorithms promise speed and efficiency gains by exploiting fine-grain parallelism and event-based computation. Examples include computational primitives, such as sorting, max, min, and median operations [70], a wide range of graph algorithms [71]–[74], NP-complete/hard problems, such as constraint satisfaction [75], boolean satisfiability [76], dynamic programming [77], and quadratic unconstrained binary optimization [78], [79], and novel Turing-complete computational frameworks, such as Stick [80] and SN P [81].

Due to the insufficient maturity of prior neuromorphic platforms, few of these proposed algorithms have been mapped to neuromorphic hardware platforms and those that have were often demonstrated in a rudimentary form with no reported speed or energy measurements. With Loihi, researchers are now able to evaluate these novel spike-based algorithms at a sufficient scale and performance to support meaningful benchmarking. This section covers examples obtained to date that have been rigorously characterized and compared to conventional solutions. So far, these examples are confirming their promise to provide orders of magnitude gains.

A. Nearest Neighbor Search

As the first demonstration of an efficient and scalable application to run on our 768-chip Pohoiki Springs system, we prototyped a temporally coded implementation of the approximate nearest neighbor search problem [16]. This implementation directly encodes search query patterns with the relative times of a single synchronized spike wavefront distributed to all Loihi chips in the system. By computing the cosine distance of the query value against all datapoints distributed over the system's cores, Pohoiki Springs is able to rapidly identify the closest matches, as defined by the angular (cosine) distance metric.

For normalized datapoints and query vectors, cosine similarity corresponds to the dot product, an operation that can be computed by integrate-and-fire neurons over time as query spikes arrive. Each stored datapoint is mapped over the input weights of a single neuron, with one eight-bit weight assigned per datapoint dimension. Given a particular broadcast input query, neurons corresponding to sufficiently close datapoints will produce output match spikes in a temporally ordered manner similar to the encoding of input query spikes, with an earlier spike indicating a stronger match. Hence, the subsequent sorting task is simplified to just observing the order in which output spikes are generated. When only the k nearest matches are needed, the network can stop as soon as k spikes are observed.

While data-parallel architectures are able to efficiently compute batched dot products using their plentiful multiply-accumulator resources operating in parallel, they are extremely inefficient at executing the sequential *top-k* sorting operation. A high-speed implementation of the *top-k* computation on a GPU requires considerable effort and extra latency and is the focus of GPU k-NN search implementations. In contrast, the operation comes nearly for free for the Loihi SNN implementation since waiting for the earliest spikes to arrive consumes no incremental energy or time.

Moreover, in contrast to the most optimized conventional approximate nearest neighbor implementations, new datapoints can be added to the search database online with $O(1)$ complexity simply by configuring more neurons in the database, which are already physically provisioned in the system.

The actual Loihi implementation uses a dimensionality reduction step, involving principal and independent component analysis (PCA/ICA), in order to support arbitrary input data types while limiting the stored datapoint dimensionality to a fixed value. The PCA/ICA reduction step also simultaneously projects the queries to a sparse representation suitable for efficient spike encoding. Both

query routing resources and synaptic memory resources scale linearly with increasing dimensionality, so a tradeoff must be made between the accuracy of the dot product calculation and the number of datapoints that can be stored in the system.

The Loihi k-NN implementation was evaluated against other state-of-the-art approximate nearest neighbor algorithms on a number of standard data sets comprising as many as 1M datapoints with 960 dimensions each. For each individual metric, it is possible to find an algorithm that outperforms Loihi, but the Loihi implementation achieves comparable accuracy while offering a unique combination of low latency (3.03 ms), high throughput (366 queries/s), low power (10.8-W dynamic neuromorphic power and 10× lower than CPU solutions), fast build time, and online insertion of new datapoints with $O(1)$ complexity. Compared to an equivalent brute force dot product implementation on a CPU, the Pohoiki Springs implementation outperforms by a factor of 685 in EDP (see [16] for details). We refer to a *k*-NN search as task 11 in Section VIII.

B. Graph Search

Our next example of computing with time is inspired by spike wavefronts observed in the hippocampus during route planning. These wavefronts appear to be part of a mental search for the shortest route through a map of the space [82]. Ponulak and Hopfield [71] proposed an algorithm that uses a spike wavefront to temporally search a graph embodied in an SNN. Synapses are modified by plasticity rules as the wavefront passes, allowing the path to later be read out of the network. Other wavefront-based algorithms have been proposed, including the classic Dijkstra algorithm, but SNN formulations, in particular, promise excellent performance by exploiting parallelism, time-based computation, and sparse spike activity.

We implemented a streamlined version of the Ponulak and Hopfield [71] algorithm, simplifying it with linear integration dynamics and binarized synaptic plasticity rules, while enhancing it with synaptic delays encoding small (6-bit) positive integer edge costs. A formal specification of this algorithm is provided in Section S.III in the Supplementary Material.

Searching a graph on Loihi involves first partitioning and mapping the graph to the physical cores of a multichip Loihi system. This host compilation process can take hours for a million-node graph, so Loihi is best suited for repeated queries on a single static graph. The source node of a search is selected by configuring its corresponding neuron to route its spikes to the host CPU. Next, the search is triggered by stimulating the destination node (neuron) to fire. This causes a wavefront of spikes to propagate through the graph until reaching the source node; at that time, the host CPU will receive a spike and halt the execution. During propagation, whenever a spike in the wavefront first reaches an intermediate neuron, plasticity rules zero the weight of the connection(s) on which it arrived, leaving behind the connection facing in the opposite direction. Once the search completes, the host CPU reads the network state, following the path of nonzero weights to discover the shortest path.

A theoretical analysis of the search phase suggests the search time scales as $O(nd\sqrt{E})$, where n refers to the number of edges along the graph's shortest path, d is the average edge cost along the path, and E is proportional to the number of edges (and nodes) in the graph. The $\sqrt{E}$ term arises as a result of the 2-D Loihi system topology and the fact that the barrier synchronization has to touch all neurons. This is a fundamental factor often overlooked in SNN performance analyses that erroneously assume that timesteps have a constant global value regardless of network scale.

The final readout phase relies on sequential von Neumann execution, which is not reflective of how such a search function would be deployed in a larger SNN application. Nevertheless, the sequential readout scales as $O(n \cdot e)$, where e is the average number of fan-out edges per node along the critical path, so, interestingly, this does not affect the asymptotic scaling behavior. Asymptotically, for large graphs, the neuromorphic algorithm, therefore, scales as $O(\sqrt{E})$, whereas even modern Dijkstra implementations that are optimized for bounded edge costs scale at best linearly in N and E [84], [85].

To assess the actual performance of Loihi's graph search implementation, we evaluated 1651 searches between randomly selected nodes in 34 Watts–Strogatz small-world graphs [83]. The graphs spanned 100 to one million nodes and 10–290 edges per node. We chose small-world graphs since they arise in many real-world settings, such as social, electrical, semantic, and logistics networks, are easily synthetically generated, and stress the communication/sorting functions of both neuromorphic and conventional search implementations. We compared the wavefront search times of Loihi to the search times of a CPU implementation[8] of Dial's algorithm, a variant of Dijkstra's algorithm optimized for bounded integer edge costs. Search time results as a function of total edges, E, are shown in Fig. 6(b).

As expected, the CPU shows an approximately linear dependence on E over the evaluated graphs. Loihi's search times initially show a sublinear dependence on E, as predicted by theory, but the maximum search times in larger graphs show an increasing dependence, closer to linear. As the likely explanation, we found that spike congestion dominates the search times, especially between Loihi chips in the larger graphs. The small graph diameters of small-world networks exacerbate this effect since most searches will typically visit the majority of all edges in the graph by the time the critical path is identified.

[8]CPU: Intel Xeon Gold 6136 with 384-GB RAM, running SLES11, evaluated with Python 3.6.3, NetworkX library augmented with an optimized graph search implementation based on Dial's algorithm. Loihi: Nahuku and Pohoiki Springs system running NxSDK 0.97.

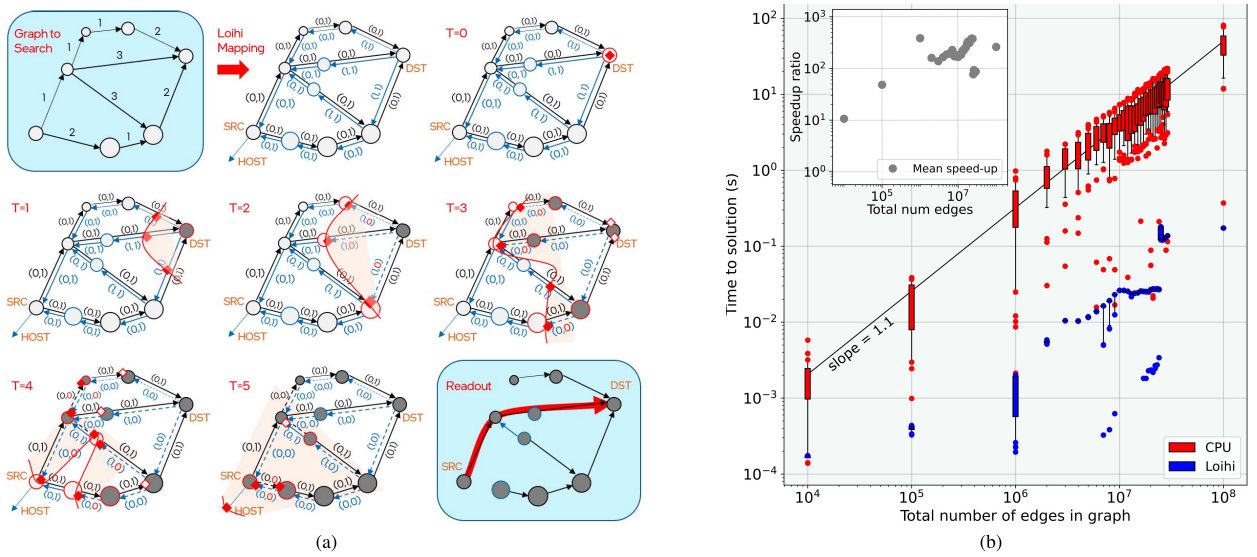


Fig. 6. ***(a) Example Loihi SNN graph search. A given directed graph is mapped to its Loihi SNN implementation, where each node becomes one or more neurons such that graph edge costs are mapped to synaptic delays, under the requirement that neurons with more than one fan-out have zero added delay. Edges are annotated as pairs of (delay and weight). During a backward-propagating search starting from the specified destination node, synaptic weights are zeroed such that the final shortest path(s) will be encoded by paths with remaining nonzero weights. (b) Graph search time comparisons for a variety of small-world graphs [83] spanning 100 to one million nodes and 10–290 edges per node, with a rewiring probability of 0.1. The main plot shows distributions of measured search times of a CPU[9] running Dijkstra's algorithm optimized for small integer edge weights, and those for Loihi to search and mark all synaptic weights along the critical path (see text for details). Depending on the size of the graph, between 1 and 102 Loihi chips are used. The outliers signify data that are below fifth and above 95th percentiles, and the boxes capture the interquartile range from Q1 to Q3. The inset shows the ratio of mean CPU search times to mean Loihi search times from the main plot. We refer to graph search as task 12 in Section VIII.***

Nevertheless, these results show Loihi outperforming the CPU by over 100× for all but the smallest graphs tested.

The shortest path search is a foundational graph algorithm, and our formulation here is representative of many similar SNN algorithms proposed in recent years that support a much broader range of graph computations [72]–[74], [77]. Loihi's encouraging quantitative results suggest that similar order-of-magnitude gains may be realized as this domain is further developed and mapped to neuromorphic hardware.

C. Stochastic Constrained Optimization

Precise spike timings can also be employed to solve the NP-complete class of constraint satisfaction problems (CSPs). A CSP involves finding permissible values to a set of variables X that satisfy a set of constraints C on the domain of allowed values of those variables. The combinatorial NP-complete nature of CSPs arises from an exponentially increasing set of possible solution configurations as the number of variables grows. Thus, finding a general solution is intractable despite the fact that verifying candidate solutions is computationally cheap.

State-of-the-art algorithms for CSPs are either systematic or stochastic greedy. In the case of systematic strategies, metaheuristics, such as backtracking, generate new candidates for X iteratively, which is guaranteed to find the complete set of solutions given enough time, but such complete solvers have exponential worst case complexity. Stochastic search strategies, in contrast, randomly draw new assignments for X resulting in an incomplete solution set and are not guaranteed to find any solutions. Nevertheless, stochastic search algorithms tend to exhibit better scalability and can also be applied to more general constraint optimization problems by designing cost functions that penalize nonsatisfied constraints.

Motivated by the early work of Hopfield and Tank [86] on solving CSPs with Boltzmann machines, neuromorphic approaches have adopted such cost function-based strategies. Jonke *et al.* [87] proposed the use of stochastic SNNs governed by an energy function

$$E = \mathbf{S}^T(t) \cdot \mathbf{W} \cdot \mathbf{S}(t) = \sum_i (S_i \cdot \sum_j W_{ij} \cdot S_j) \qquad (1)$$

to solve CSPs. In (1), $\mathbf{S}$ is the instantaneous spike vector and $\mathbf{W}$ the synaptic weight matrix. In contrast to Boltzmann machines, $\mathbf{W}$ can be either symmetric or asymmetric. The SNN is set up in such a way that different values of the CSP variables X are represented by one-hot coded WTA subnetworks, and $\mathbf{W}$ encodes the constraints C. Buesing *et al.* [75] had shown that the stochastic dynamics of such networks converge exponentially fast to a probability distribution $p \propto \exp(-E/\eta)$, where η parameterizes the degree of stochasticity. In addition, the fine-scale timing dynamics of SNNs allow them to readily escape from local minima, making them more effective in finding the

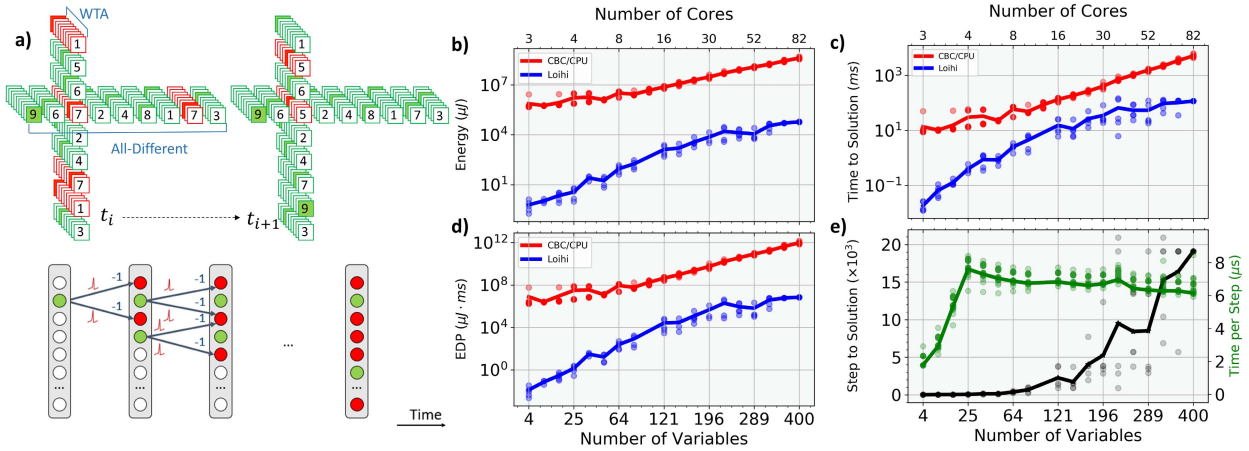


Fig. 7. ***(a) Top: configuration of multiple WTA networks participating in one exemplary row/column constraint of the Latin square problem during a state transition from time t_i to t_{i+1} that reduces the number of conflicting variable assignments. Each variable with a legal (green) or conflicting (red) assignment is represented by a WTA network. Bottom: illustration of iterative search space pruning. As neurons fire (green), they inhibit other conflicting neurons (red). (b)–(d) Average energy, time, and EDP scaling performance over a range of 4–400 variables of the Loihi CSP solver compared to CBC algorithm on a CPU[11]. For each problem size, the average is over five random initialization of the compartments voltage shown as data points. In the case of the 20×20 problem $E_{CPU}/E_{Loihi} = 2.79 \times 10^3$, $T_{CPU}/T_{Loihi} = 4.39 \times 10^1$ and $EDP_{CPU}/EDP_{Loihi} = 1.22 \times 10^5$. (e) Decomposition of the time to solution T into the number of time steps to solution t (left) and mean time per timestep τ (right). We refer to CSP as task 13 in Section VIII.***

global minimum than Boltzmann machines, even though both sample from the same underlying distribution. For an accessible overview of how stochastic spiking neurons solve such problems, see [88].

For a neuromorphic CSP solver to be viable, the SNN must not only visit energetically minimum states but also detect when one is visited. Previous neuromorphic hardware implementations [89]–[96] required a von Neumann processor to continuously read out the entire high-dimensional network state **S** to evaluate the cost function and identify solutions. This leads to an impractical off-chip communication bottleneck that only worsens with increasing problem size. Instead, our solver (see Section S.IV in the Supplementary Material) also computes the cost function in a distributed and event-based way in the neuromorphic network. It only communicates with a CPU whenever a solution is found.

We use the NP-complete Latin square problem[9] to demonstrate and assess the performance of our distributed Loihi CSP solver. Fig. 7(a) depicts the principle of operation of the solver. WTAs represent individual variables X in the Latin square problem, with the ith neuron in each WTA corresponding to the variable assuming value $i \in (1, \ldots, N)$. The *all-different* constraints between variables in a row or column are realized by inhibitory connections between WTA neurons corresponding to conflicting variable assignments. As a result of the stochastic network dynamics, some neurons will fire earlier than others and inhibit other conflicting neurons whose activity would contradict the current state. This process results in an iterative pruning of the search space until, finally, only nonconflicting neurons remain active, corresponding to the solution of the Latin square problem.

Following this approach, we have benchmarked the time and energy required by Loihi to find solutions against those of a mixed-integer linear programming solver on a CPU.[10] We chose the Coin-or Branch and Cut (CBC) solver provided by COIN-OR projects[11] since, like the Loihi solver, it uses an incomplete energy minimizer and is among the best performing open-source linear programming solvers in a range of benchmarks [97].

The results in Fig. 7(b)–(d) illustrate that the Loihi solver is significantly faster and more energy-efficient than the CPU reference. Overall, it achieves at least three orders of magnitude lower energy–delay product over a wide range of problem sizes spanning from four to 400 CSP variables. Besides Loihi's energy efficiency, the key reason behind Loihi's outperformance in this domain is the scaling behavior of the time to solution $T = t \cdot \tau$, which is composed of the algorithmic timesteps to solution t and the average time per timestep τ, as shown in Fig. 7(e). The initial rise of T in the single-core regime up to a problem with 25 variables is mainly driven by τ. However, as the problem size grows further across multiple cores, τ flattens out favorably below 8 μs per timestep due to multicore parallelism. Beyond 25 variables, a continued exponential rise of T driven by t is expected as an unavoidable consequence of the exponentially growing search space.

Although, like any incomplete solver, our solver is not able to formally guarantee the existence of a solution or to always find any or all solutions, in practice, it has found optimal solutions for the largest CSPs to date of any

[9]A Latin square consists of an $N \times N$ array of variables, each of which can take on N possible values such that no number is repeated in a row or column of the grid.

[10]Loihi: Nahuku board running NxSDK 0.95; CPU: Intel Core i7-9700K; and RAM: 128 GB, running Ubuntu 16.04.6 LTS.

[11]www.coin-or.org/projects/

neuromorphic system without yet exhausting the resources of a single Loihi chip. Furthermore, its ability to find solutions incrementally makes it particularly attractive for time-critical applications. Relaxing the notification threshold of the online energy evaluation network causes the solver to more quickly identify approximate solutions, which provides a latency–accuracy tradeoff mechanism that could be valuable for some applications.

Loihi has also been used to solve other types of CSP problems, such as assigning optimal asset allocations in the context of real-time decision making [98], achieving 1000× faster execution than an exact algorithm on GPU or to tackle Boolean satisfiability problems [76] (see Section S.IV in the Supplementary Material).

These results illustrate how stochastic spike timing dynamics can expand the space of computation supported by neuromorphic networks, yielding surprisingly fast and efficient results. This domain is still in its infancy, with much still to be learned. For example, a better understanding of the role that noise correlations play in our CSP solver may unlock even greater gains, as discussed in Section S.IV in the Supplementary Material. Furthermore, stochastic SNNs show promise for a number of challenging computational problems beyond constraint satisfaction and optimization. Applying methods from probabilistic signal processing, stochastic SNNs can be designed to tackle a variety of inference and learning problems that optimally leverage their time-coding capabilities [99]. Stochastic SNNs have been shown to perform probabilistic inference on general graphical models through spike-based stochastic sampling and can even learn such models using unsupervised synaptic plasticity mechanisms [88]. These algorithms are conventionally expensive to compute and, therefore, stand to benefit substantially once fully formulated in this neuromorphic paradigm.

VI. APPLICATIONS

Beyond algorithmic benchmarking, a number of promising applications have been demonstrated on Loihi, as described here.

A. Event-Based Sensing and Perception

Event-based sensing is a rapidly evolving sister technology to neuromorphic computing. Event-based vision sensor pixels each detect intensity changes over time, asynchronously generating events whenever the change exceeds a threshold. They exhibit an impressive combination of properties, including self-adaptation, low-power consumption, low-latency, and high dynamic range [100]. Unfortunately, their spiking output differs so significantly from traditional computer vision frames that new processing algorithms and architectures must be developed in order to realize compelling real-world applications.

Great early strides have been made on extracting useful information from spiking vision data (recently reviewed in [101]). However, these algorithms are still nascent. Many can only handle low-resolution data, and they often rely on offline or frame-based processing pipelines that negate the desirable low-power and low-latency characteristics of the sensors themselves.

Architectures such as Loihi preserve these characteristics by natively operating on spiking data in an event-based fashion. Much work remains to be done on the spiking algorithm front, but steady progress is being made. Early efforts demonstrate digit recognition, fusion of visual and tactile perception [40], fusion of visual and EMG information [39], persistent attention and tracking [68], and online learning of gestures [102] using event-based sensors interfaced to Loihi.

As the resolution of event-based sensors continues to scale, we require computing architectures that can scale with them, and scalable parallel computing architecture, such as Loihi, is a natural choice. However, increasing resolution also increases the communication bandwidth requirement between sensor and processor, which, in turn, introduces a number of challenges relating to power, interface cost, and degraded temporal resolution. These motivate a fresh look at the partitioning of neuromorphic processing across sensor and computing elements, a topic of ongoing research (see Section VII-C).

B. Odor Recognition and Learning

Olfaction is another emerging sensing domain showing promise for Loihi. Although odor and chemical sensing may not obviously benefit from spike-based low-latency processing, odor sensing poses its own technical challenges that make it a good match for neuromorphic technology. Today's odor sensors, just like odor sensing neurons, are unreliable and require frequent recalibration. High levels of noise and occlusion inherent to this modality create a difficult and potentially compute-intensive recognition problem at odds with edge device deployment. A large diversity of real-world odors with significant natural variability calls for online learning and fine-tuning in the field.

At the same time, chemical sensing is one of nature's oldest sensing modalities. The neural circuits found in the mammalian olfactory bulb and the insect antenna lobe are examples of convergent evolution, having independently evolved over hundreds of millions of years to solve the same problem in remarkably similar ways [103]. This suggests that these circuits are specially optimized for this task, and studying them may yield new ideas for machine learning. In fact, neuroscience in this domain is relatively mature and motivates a bottom-up approach to algorithm discovery.

Harvesting insights from recent neuroscience modeling, Imam and Cleland [104] abstracted a biophysical model of the olfactory bulb to a level that could be mapped into the Loihi architecture. They implemented mechanisms supporting both odor recognition and new odor learning and evaluated the model's performance using a publicly

available gas sensor array data set [105]. The model invokes many features that are uncommon in conventional machine learning, such as phasor attractor dynamics, spike phase coding of information, random high-dimensional projection through recurrent neuron populations, neurogenesis, and fast-acting learning rules mixing both delay and weight adaptation. While a conventional formulation of the algorithm can surely be written to run on a von Neumann processor, it would be more difficult to formulate and inefficient to execute.

Given single training samples over ten classes of chemicals, Loihi's neuroscience-inspired algorithm is able to successfully classify test samples drawn from the same data set or corrupted with high levels of impulse noise. Loihi achieves a high level of classification accuracy (92%), outperforming by over 40% four other conventional algorithms, including a seven-layer backprop-trained deep autoencoder with the same number of units as the Loihi model. Furthermore, the Loihi algorithm is able to learn new odors sequentially with no discernable degradation in classification performance on odor classes learned earlier, whereas, with a similar training sequence, the deep autoencoder's performance drops to chance level. Given a sufficient number of training samples (3000 per class), the deep autoencoder is able to reach the same 92% classification accuracy as Loihi achieves with a single sample, and given a further doubling of training samples (to 6000 per class), it can outperform Loihi by 4%. Scaled over a wide range of network sizes from 20 cores to 128, Loihi is able to classify odor samples in under 3 ms with less than 1 mJ of energy [104].

C. Closed-Loop Control for Robotics

Closing the loop between sensing and actuation is another exciting area for neuromorphic computing and Loihi. Event-driven processing in neuromorphic hardware matches the temporal character and low-latency requirements of closed-loop control. Several approaches to motor control have been demonstrated on Loihi.

The NEF and Nengo [20] have been used to configure an adaptive neural implementation of a proportional–integral–differential (PID) controller, in which Loihi's learning features allow the integral (I) term to adapt online to mitigate state-dependent perturbations. The latter was demonstrated using a simulated SNN [106] and was recently realized on Loihi to control a six-degree-of-freedom force-controlled robot arm [50]. This adaptive PID controller on Loihi outperforms CPU and GPU implementations [50]. The CPU and GPU use 4.6× and 43.2× more power than Loihi, respectively. The CPU and Loihi had similar latency, but the GPU was 42% slower. Fast processing leads to faster convergence to the set value and, thus, more precise control. The lower Loihi latency also resulted in 1.49× and 1.57× improved accuracy over the CPU and GPU when subjected to a model of accelerated frictional wear.

Glatz *et al.* [107] proposed another SNN controller design, later implemented on Loihi by Stagsted *et al.* [108] to control a one-degree-of-freedom drone platform. Demonstrating the benefits of event-based vision, the drone processes input from a DAVIS240C event-based camera to visually track the high-speed roll motions of an artificial horizon drawn on a spinning disk [109] (see Fig. S5 in the Supplementary Material). An earlier implementation used a conventional embedded CPU to process the event input. On each event arrival, it computed the horizon angle with a Hough transform, which was then passed to a PID controller to generate motor control signals that actively drive the horizon angle to zero. The Loihi implementation improves on that earlier result by replacing the CPU with an end-to-end event-based SNN. It implements the Hough transform with a prestructured four-layer SNN followed by an event-based PID controller implemented with DNF attractor networks. DNF populations store the controller's measured and set values that are combined through relational connections to compute the controller's error, derivative, and integral terms. The Loihi implementation supports control rates of up to 20 kHz with less than 1-ms latency [110], improving the earlier CPU-based control rate by 22× and its latency by 15%. Moreover, while the CPU implementation offers no efficient path to supporting more complex visual processing, the Loihi implementation can easily integrate other event-based inference networks, such as backprop-trained convolutional SNNs, to greatly expand its onboard visual processing intelligence while maintaining low latency and power.

Another example of neuromorphic motor control uses central pattern generators (CPGs), inspired by neural circuits generating rhythmic activity patterns in animals [111]. Several groups have demonstrated their capability to generate locomotion patterns of insect-like robots [112]. Recently, Polykretis *et al.* [113] realized a CPG on Loihi to control the gait of a hexapod robot. They configured a circuit of spiking neurons to exhibit bursting behavior, connected these bursting circuits into oscillating pairs, and used their activity to generate gaits for the robot. Using a similar approach of mimicking biological circuits, Balachandar and Michmizos [114] demonstrated precise (error $< 3^{\circ}$) real-time control of a robotic head, inspired by oculomotor control in animals. CPG-based control can be an important component of the overall control architecture, in which complex rhythmic patterns need to be generated in a parametric and adaptive way, as, e.g., in robots that change their locomotive behavior when switching between environments (water versus ground) [115]. This is an active area of research both in bioinspired robotics and neuromorphic computing [116]–[119].

D. Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is an important robotics task that requires: 1) *state estimation*

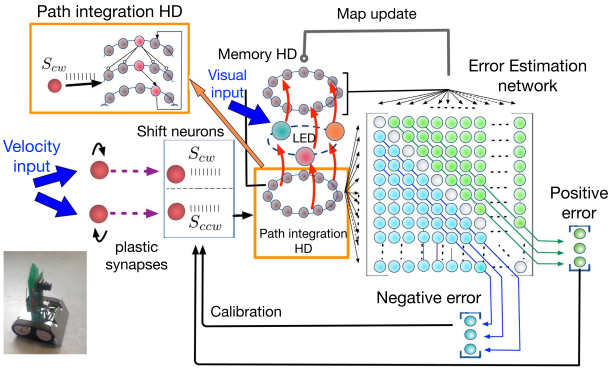


Fig. 8. ***Neuromorphic SLAM SNN: the velocity signal, scaled with plastic synapses to the shift neurons, drives the path integration network, in which an attractor-bump forms over current HD. When a visual cue (a blinking LED) is detected, plastic synapses store associations between the visually driven LED neurons and the HD rings. When the visual landmark is revisited, an error between the current HD estimate and the previously learned one is computed. Depending on the sign and magnitude of the error, the learned map is updated, and the velocity representation is calibrated.***

from onboard sensory information, often using sensor fusion and path integration, to maintain the absolute position of the agent and 2) *formation of a map*, often by storing locations of objects of interest in the environment. Because of errors and drift in the state estimation, the key problem in SLAM is to detect and reduce errors when forming the map. The problem is typically formulated as optimization and becomes a prohibitively costly computation in large environments.

Neural network solutions to the SLAM problem were inspired by biological circuits, known from neurophysiological studies of animal navigation. The first such SLAM network, RatSLAM, was proposed by Milford *et al.* [120]. Functional modules of this network, such as head direction, speed, place, and visual landmark cells, have been implemented on Loihi on a simple mobile robot platform, showing how precise state estimation can be achieved in a neuromorphic chip and how maps can be formed using on-chip synaptic plasticity [121].

In this work, DNFs (see Section IV) are used to represent the heading direction (HD) and position of the robot. The 1-D HD "ring" (see Fig. 8) integrates information from different sources. If HD cannot be directly sensed, dead reckoning based on path integration of motor commands or IMU velocity measurements can be used instead. Precise path integration is achieved by using velocity measurements to drive a set of rate-coded velocity neurons ("shift neurons" in Fig. 8). A stronger input results in a higher firing rate of the corresponding velocity neuron and a faster shift of the attractor-bump in the HD ring. As shown in [122], this velocity mapping can be precisely calibrated, achieving an error below 1°.

Drift in the pose representation is corrected when the robot recognizes a location that it has previously visited. To achieve pose correction in an SNN, positions in the environment are associated with visual cues, using plastic synapses on Loihi, shown as red arrows in Fig. 8. Whenever a visual cue learned in this way is recognized later, the pose is recalled. The competitive dynamics of the DNF storing the HD inhibit the current path-integrated estimation of the pose and allow it to be replaced with the memory-induced estimation. Synaptic potentiation and depression lead to online updates of the map.

In a similar line of work, Tang *et al.* [123] realized a 1-D SLAM network on Loihi that performed head direction estimation using spike-based recursive Bayesian inference and map formation based on perceived distances to objects. In this SNN, also inspired by RatSLAM, a DNF performs path integration, while a network of multicompartment neurons combines the dead reckoning estimate with the perceived one in an approximately Bayesian way. A network for reference frame transformation forms an allocentric representation of the environment in another DNF, which uses synaptic plasticity rules to learn and update the map. This SLAM model was benchmarked on Loihi against a standard particle-filter-based SLAM implementation, GMapping, running on a CPU (Intel i7-4850HQ). Loihi was found to consume 100× less dynamic power than the CPU while achieving similar accuracy (within five degrees of angular precision). We refer to 1-D SLAM as task 10 in Section VIII.

E. Other Applications

A number of other early stage application demonstrations with Loihi have been published. Some recent examples include a feedforward associative memory with the ability to learn new patterns online [124], a dynamic radio frequency (RF) waveform adaptation algorithm for robust communication in noisy RF environments [125], and an event-driven random walk-based solver for the heat diffusion equation running on both Loihi and TrueNorth [126].

Many other application domains are being actively pursued without any published results to date. These include anomaly detection for security and fault monitoring, sparse coding and reconstruction applications, particle collider trajectory classification, brain-computer interfacing with EEG sensors and direct neural probes, drone-based structural health monitoring, and numerous audio applications, such as low-power keyword spotting, speech recognition, speaker identification, denoising, and sound localization.

VII. OUTLOOK

A. Deep Networks

Our results in Section III show that ANNs converted to rate-coded deep SNNs on Loihi may offer significant gains in energy efficiency compared to ANNs on conventional architectures but generally result in long latencies, especially for large-scale problems spanning multiple chips. Interchip mesh congestion is a problem in these networks. The congestion can be mitigated by introducing

relay cores or multicast interchip links in future silicon to perform spike fan-out locally on the destination chip, but, even with such measures, the latency versus precision tradeoff remains for these networks and worsens with deeper networks. These attributes render converted rate-coded models less attractive for neuromorphic architectures optimized for sparsity and, therefore, demand other approaches.

At the algorithmic level, direct training approaches have already demonstrated vast improvements in latency and, thereby, also indirectly in energy. When combined with conversion approaches, hybrid direct approaches promise to reduce congestion and latency. Wu *et al.* [127] proposed the TANDEM framework for cotraining of an ANN and SNN, while Rathi *et al.* [32] developed a hybrid SNN training approach, in which backpropagation is used to train the network further after conversion. Both approaches result in inference times on the order of 10× shorter compared to standard rate coded networks. Another direction is to use temporal instead of rate-based spike coding to represent ANN activities. Zhang *et al.* [128] have shown that temporal coding using a logarithmic time scale is the most efficient, second only to time-to-first-spike (TTFS) [129] coding. However, compared to TTFS, the logarithmic time scale approach is more compatible with popular deep learning techniques.

Various forms of network compression developed recently [130] save both energy, time, and memory resources that are especially precious for compute-in-memory architectures. These include dimensionality reduction techniques and opportunities to exploit sparsity arising from pruning before, during, or after offline training. In addition, algorithms such as deep rewiring [36], which keeps the level of sparsity stable and constantly recycles less important synapses to where they are most needed for a task, lend themselves well for memory-constrained, adaptive online learning approaches on neuromorphic systems.

B. Online Learning

Today, online backprop approximation algorithms, such as SOEL and PES (see Section III-C), provide valuable examples that work within the constraints of the Loihi architecture. Longer term, more general algorithms, such as eligibility propagation [47] and equilibrium propagation [131] that approximate BPTT without the anticausal requirement of propagating information back in time, may succeed in scaling up online learning. These neuromorphic approaches face hardware efficiency and convergence challenges, given that they process training data sample-by-sample rather than in batches, and their weight updates throughout the network tend to be nonsparse. Fundamentally, all online backprop approximation algorithms suffer from the same basic challenge that network parameters are updated in small gradient-directed steps and require large numbers of supervised training samples to generalize.

Achieving supervised real-time online learning represents a significant challenge to the field of artificial intelligence in general, and we are still very far away from realizing it in deep networks. New ideas are needed to achieve continual learning from sequential data samples that do not naturally arrive independently and identically distributed.

Looking to nature as a guide, as well as to the examples running on Loihi today, we foresee a greater focus on network modularity and *shallow learning* algorithms that form associations between different neuron populations and between distributed semantic representations. We view backpropagation primarily as a tool for offline training using large precollected data sets, with a diversity of fast-acting shallow plasticity rules providing rapid online learning and adaptation.

C. Sensor Integration

Loihi's low latency is particularly valuable for sensory processing where the value of sensory information degrades rapidly over time. However, as the algorithms and applications for sensory processing mature and scale, they will demand more of the architecture and interfaces. In order to move beyond the relatively low-dimensional data streams featured in the examples of this survey, it would not be enough to simply boost the bandwidth of chip interfaces and spike encoding functions. Neuromorphic chips are fundamentally incompatible with standard data formats, which tend to come in dense arrays captured on synchronous intervals, i.e., the legacy of decades of conventional computing. To truly unlock the value of neuromorphic computing at scale, offering compelling power and latency advantages for all manner of computing devices processing real-world data streams, the sensors themselves will need to be rearchitected in an event-based paradigm. They will need to perform sufficient sparse feature coding to avoid saturating low-power interface bandwidths with irrelevant or unchanging raw data that only degrade timing resolution and waste downstream processing power.

Accordingly, we see a need for tighter integration of sensors and processing, motivating disruptive codevelopment across neuromorphic sensing circuits, architecture, and algorithms. For example, the PCA/ICA preprocessing performed on the host CPU in our k-NN example (see Section V-A) suggests a simple linear transform that could be embedded in any sensor to sparsify its output data. LASSO-style sparse coding offers another technique that might conventionally be seen as too heavyweight for sensor integration but, in its neuromorphic LCA form, comes nearly for free in power and latency. Following nature's lead, we foresee more exotic nonlinear transformations providing even greater gains, such as spectral transforms with Hopf oscillator cascades, as in the cochlea [132], and motion detection or segmentation using synchronization effects as some have modeled in the retina [133].

Vision applications are likely to be the first to demand such tighter integration. Event-based cameras and spike-based neuromorphic processing are two parallel architectures communicating over a serial interface, and raw vision data require particularly high bandwidth. The 3-D vertical integration offers the potential to drastically reduce the power and latency of the interface and allow for a tileable sensory processing architecture. From an engineering perspective, the design and fabrication of this interface remain open challenges, but the required technologies are already maturing in conventional vision sensors.

D. Robotics

For many decades, technologists and science fiction authors alike have foreseen robots that are able to navigate and interact in the real world, operating with autonomy and agility alongside humans. Such robots remain out of reach today although great strides are being made on some of the required sensing, actuation, mechanical, and energy storage technologies. To intelligently control these robots of the future, a difficult integration must occur between classical control theory that relies on precise models of the environment and artificial intelligence that would ground such models in perception. Interacting with dynamic and often unpredictable real-world environments remains challenging for the most advanced modern robots, but it comes effortlessly to humans. It is exactly the task that biological brains have evolved to solve, and it is one of the most promising application areas of neuromorphic technology.

Deep neural networks have enabled state-of-the-art results in computer vision and are the natural first choice when building powerful perceptual systems for robots. Despite excellent results in constrained environments where sufficient training data can be precollected, deep learning approaches still fall short of addressing the needs of real-world robotic vision even for tasks such as object recognition that are commonly considered solved [134]. While today's neuromorphic technology might lower the power and latency of DNNs for certain visual inference tasks, new adaptive algorithms are needed that can cope with the full variability and unpredictability of the real world. We believe that these will be well matched for the unique features of neuromorphic architectures and will come in time with ongoing algorithm-hardware codevelopment.

Beyond basic visual inference, robust robotic systems will need to integrate multiple sensing modalities: vision, sound, proprioception, and touch. Loihi has shown early promising results for combined tactile and visual sensing [40], showing how event-based data coding and processing can provide a unifying language for efficient multisensory integration, with spikes encoding both temporal and spatial information of significance to the task across modalities. The dynamical nature of certain neuromorphic networks, such as DNFs (see Section IV-B), provides another unifying primitive, in which memory states can be created in attractor networks to bridge the different timescales of diverse sensory streams. Furthermore, these neuromorphic networks with recurrent and feedback connections enable top-down, attentional modulation of sensory processing, focusing computing resources on the most relevant aspects for the task while helping to ignore noise and occlusions affecting input from any one modality.

Demonstrating progress toward the above goals, we recently integrated multiple neuromorphic sensorymotor networks into a single Loihi chip to control a humanoid robot's interaction with its environment in an object-learning task.[12] In this work, three SNNs were implemented on Loihi: an object-recognition network receiving input from an event-based camera and labels extracted from speech commands; a spatial-memory network that tracked the pose of the robot's head from its motor encoders and memorized object locations using on-chip plasticity; and a DNF-based neural state machine responsible for reconfiguring different parts of the architecture as required by the current behavior (looking, learning an object, recognizing it, or communicating with the user). While we are still far from implementing the robust, adaptive brains that future robotic systems will require to interact freely in the real world, this work shows that we can already build relatively complex neuromorphic applications by composing heterogeneous modules drawing from a toolbox of common algorithmic primitives, such as spike-based communication, attractor networks, top-down and bottom-up attention, working memory, and local learning rules.

E. Planning, Optimization, and Reasoning

Planning and reasoning are arguably the most advanced and elusive capabilities of natural intelligent systems. While recent progress in deep learning has provided great gains in subsymbolic inference and learning, neural networks have yet to offer the same gains for higher order symbolic and analogical reasoning tasks, and let alone provide a unified system that can leverage these capabilities to plan actions and optimize for high-level objectives. Vector symbolic architectures (VSAs) offer a mathematical, connectionist framework that supports rich knowledge representations and reasoning in high-dimensional spaces [135]. VSAs interfaced with deep networks and generalizations of the optimization and search algorithms described in this survey could provide a path to enabling fast, efficient, and scalable next-generation AI capabilities on neuromorphic hardware.

The first step toward a scalable VSA framework was recently implemented on Loihi in the form of a spike-based phasor associative memory network, TPAM [136]. Associative memories in high-dimensional VSA representations

[12]See https://vimeo.com/intelpr/review/486107679/6863af9df8

require a cleanup operation that relies on all-to-all communication across vector dimensions, limiting VSA's scaling and efficiency on any hardware architecture. The TPAM, in contrast, and especially Loihi's spike-based TPAM implementation, introduces sparsity in both the connectivity and activity of the network, thereby increasing its ability to scale. The phasor-based dynamics of TPAM also provide other novel benefits, such as the ability to reliably store and operate on graded-valued patterns, unlike traditional Hopfield networks that can only reliably store binary-valued patterns, even when implemented with graded-value rate neurons.

A future fully developed spike-based phasor VSA framework promises sufficient capacity to solve nontrivial AI problems quickly and efficiently. In scaled form, emerging VSA algorithms, such as the Resonator for vector factorization [136], [137], could provide breakthroughs for AI problems that are poorly solved today, such as scene understanding and hierarchical semantic decomposition.

F. Programming Model

With Loihi's promising results and the broad space of possibility that these open up, soon, one of the most pressing problems facing the neuromorphic field will be its fragmented and noncomposable collection of programming models and frameworks. While a number of SNN development frameworks have been released for use, they all generally fall into one of three categories: point tools for optimizing SNN parameters with supervised training, usually with deep learning techniques (SNN Conversion Toolbox [21], SLAYER [22], Whetstone [138], and EONS [139]), SNN simulators for conventional architectures that offer low-level programming APIs (Brian 2 [140], BindsNET [141]), or low-level interfaces and runtime frameworks for configuring neuromorphic hardware (PyNN [142], Fugu [143], and our own NxSDK). While the increasing level of exploration and activity in this space is encouraging, none of these frameworks yet present compelling new programming abstractions that are composable and span a wide diversity of algorithms understudy in the field, such as those that are covered by the examples in this survey.

One possible exception is Nengo [20], a toolchain built around its foundational NEF [19] that optimizes the parameters of spiking neuron populations to implement defined dynamical system equations. Nengo has proven to be successful at composing a particular class of rate-coded SNNs, including deep networks, to construct large-scale applications with promising functionality in simulation, for example, the SPAUN model [144]. Nengo supports a number of conventional and neuromorphic execution platforms, including Loihi through its back-end interface to NxSDK. Several applications coded in Nengo run today on Loihi and have been covered in this survey [28], [30], [50]. Importantly, members of the neuromorphic community who are not Nengo developers have successfully used the toolchain to develop their own applications and run them on Loihi [124], [125].

Based on the perspectives that emerge from our survey of Loihi results, Nengo provides a good first step toward a productive development framework, but a fundamental expansion of the abstractions and compositional tools will be needed to encompass the wider scope of computation that neuromorphic networks support, especially the stochastic, temporally coded, and directly trained networks that have demonstrated the greatest quantitative gains in latency and efficiency.

Without such a broader framework utilizing new abstractions, many of the compelling examples characterized on Loihi will remain isolated studies showing promise but unable to connect to other modules and enable functionality that is at least equal to the sum of their parts, if not greater.

G. Economic Viability

The tight integration of memory and computation in neuromorphic architectures is both its blessing and its curse. The elimination of the von Neumann bottleneck frees the hardware implementation to splinter into an arbitrarily granular collection of tiny processing elements without any need for a monolithic memory. This is optimal for energy efficiency and sparsity but is potentially devastating for overall system cost.

The overall economics of today's semiconductor industry are highly optimized for the von Neumann architecture. The per-bit cost of monolithic dynamic random access memory (DRAM) is on the order of 100× cheaper than the per-bit cost of the densest memory available in a logic process. For large-scale memory-bound workloads, conventional architectures achieve a cost optimum by partitioning their physical implementation between cheap DRAM and expensive logic. A neuromorphic implementation must choose one or the other, and architecture as complex as Loihi's, with over half of its silicon area devoted to logic circuits, can only be implemented in a logic process today. Hence, scaling up Loihi's architecture comes at the cost of expensive logic state bits.

For large-scale applications, this inevitably puts neuromorphic technology into a high-end niche. To broaden this niche, neuromorphic technology needs to add value for small-scale problems, which suggests that its first commercially viable applications will emerge at the edge and in tightly integrated sensors and robotic systems.

Long-term, fundamental manufacturing technology innovations will be needed to reduce the cost of neuromorphic architectures. This could come from dense and cheaply integrated emerging memories, such as crossbars of resistive, magnetic, or phase change devices. Further cost gains could come from storing multiple bits per device, although a truly analog memory needs to maintain its density and cost advantages without forcing too much of the surrounding architectural elements into the analog domain, which would introduce its own cost challenges.

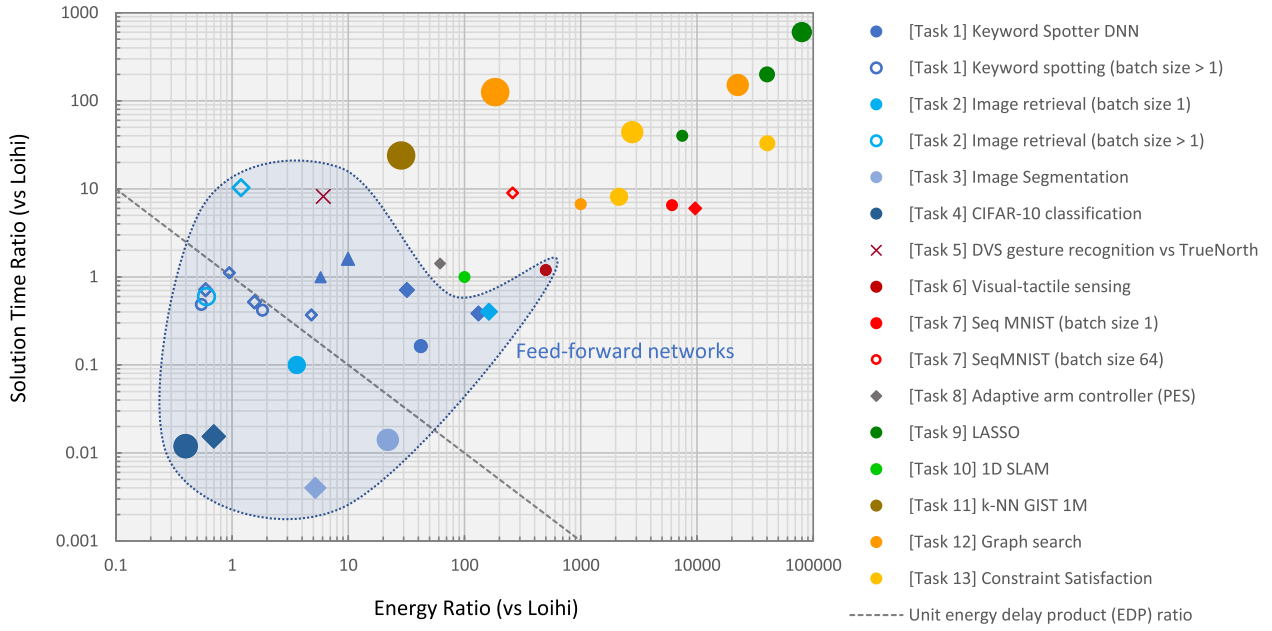


Fig. 9. ***Unified energy and delay ratio plot over a selection of benchmark workloads from this survey. Each point corresponds to a comparison of a particular workload running on Loihi versus the equivalent workload on a reference architecture. All comparisons have been controlled for accuracy, and Loihi's accuracy is approximately equal or better than the reference solution precision. The class of reference architecture is indicated by the marker shape: Circles indicate Intel Core or Xeon CPUs; diamonds indicate Nvidia GPUs; triangles indicate Intel Movidius Neural Compute Stick; and crosses indicate IBM's TrueNorth architecture. All workloads correspond to batch size 1 processing, except for open markers that indicate batched data configurations (all workloads on Loihi are batch size 1). Marker color indicates the class of network: Blue: feedforward rate-coded networks; red/orange: backpropagation-trained; gray: feedforward networks with online learning; green: attractor networks; and yellow: SNNs exploiting temporal coding or precise spike timing. The size of the datapoints represents the complexity of a task as measured by the required number of cores on Loihi (logarithmic scale).***

Without breakthroughs in the density of low energy, high-speed CMOS-integrated memory, the cost will limit the proliferation of all but small-scale neuromorphic technology into mainstream devices—even once the architecture, algorithms, and software challenges are resolved.

VIII. CONCLUSION

Fig. 9 presents many of the latency and energy benchmarking results described in this survey in a unified view. This 2-D plot emphasizes the two primary benefits offered by Loihi and event-based neuromorphic architectures, in general, compared to today's commercially available programmable architectures. The diagonal dashed line in this chart represents the energy-delay ratio parity line; comparison points to the bottom left of this line indicate the reference architectures outperforming Loihi, whereas points further to the top right of that line indicate better performance by Loihi.

Clear trends emerge from this unified view. Feedforward networks and especially rate-coded feedforward networks are the least compelling match for Loihi, especially larger networks that perform significantly worse on Loihi. Great gains are possible when spike timing becomes computationally significant, whether through supervised optimization using backpropagation or through analytical design. All of the best performing workloads on Loihi make use of highly recurrent networks.

Other evaluation dimensions, such as throughput when processing batched data, maximum achievable accuracy, and silicon cost per function, in general, show a less favorable view for Loihi. Nevertheless, it is clear that, for a particular expanding collection of workloads, the particular scope of which is still ongoing research, neuromorphic architectures can provide gains measured not in percentage points but in factors or even orders of magnitude. ■

Acknowledgment

This article reflects dedicated efforts over several years from past and present members of Intel's Neuromorphic Computing Lab, spanning silicon to software and systems engineering. Of particular note for their contributions: Harry Liu, Nabil Imam, Bodo Rueckauer, Connor Bybee, E. Paxon Frady, and Fritz Sommer. The authors also thank many PIs and especially the students of the Intel Neuromorphic Research Community for their admirable patience and persistence in working with our research silicon and tool flow. They also thank the Applied Brain Research Team, Sumit Bam Shrestha, Konstantinos Michmizos, Guangzhi Tang, Benjamin Tee, Wolfgang Maass, and Alex Kass' Emerging Technologies Team at Accenture Labs.

REFERENCES

[1] J. C. Hay, B. E. Lynch, and D. R. Smith, *Mark I Perceptron Operators' Manual*. Buffalo, NY, USA: Cornell Aeronautical Laboratory, Jul. 1960.

[2] E. Painkras *et al.*, "SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation," *IEEE J. Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, Aug. 2013.

[3] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2010, pp. 1947–1950.

[4] H. Markram, "Seven challenges for neuroscience," *Funct. Neurol.*, vol. 28, no. 3, p. 145, 2013.

[5] O. Rhodes *et al.*, "Real-time cortical simulation on neuromorphic hardware," *Phil. Trans. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 378, no. 2164, Feb. 2020, Art. no. 20190160.

[6] T. Wunderlich *et al.*, "Demonstrating advantages of neuromorphic computation: A pilot study," *Frontiers Neurosci.*, vol. 13, p. 260, Mar. 2019.

[7] J. C. Knight and T. Nowotny, "GPUs outperform current HPC and neuromorphic solutions in terms of speed and energy when simulating a highly-connected cortical model," *Frontiers Neurosci.*, vol. 12, p. 941, Dec. 2018.

[8] C. A. Mead and M. A. Mahowald, "A silicon model of early visual processing," *Neural Netw.*, vol. 1, no. 1, pp. 91–97, Jan. 1988.

[9] B. V. Benjamin *et al.*, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proc. IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[10] N. Qiao *et al.*, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses," *Frontiers Neurosci.*, vol. 9, p. 141, Apr. 2015.

[11] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, Aug. 2014.

[12] K. L. Fair, D. R. Mendat, A. G. Andreou, C. J. Rozell, J. Romberg, and D. V. Anderson, "Sparse coding using the locally competitive algorithm on the TrueNorth neurosynaptic system," *Frontiers Neurosci.*, vol. 13, p. 754, Jul. 2019.

[13] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.

[14] M. Davies, "Benchmarks for progress in neuromorphic computing," *Nature Mach. Intell.*, vol. 1, no. 9, pp. 386–388, Sep. 2019.

[15] C.-K. Lin *et al.*, "Programming spiking neural networks on Intel's Loihi," *Computer*, vol. 51, no. 3, pp. 52–61, Mar. 2018.

[16] E. P. Frady *et al.*, "Neuromorphic nearest neighbor search using Intel's Pohoiki springs," in *Proc. Neuro-Inspired Comput. Elements Workshop*, Mar. 2020, pp. 1–10.

[17] K. A. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 47, no. 5, pp. 416–434, May 2000.

[18] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240 × 180 130 dB 3 μs latency global shutter spatiotemporal vision sensor," *IEEE J. Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, Oct. 2014.

[19] C. Eliasmith and C. H. Anderson, *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems* (Computational Neuroscience Series). Cambridge, MA, USA: MIT Press, 2002.

[20] T. Bekolay *et al.*, "Nengo: A Python tool for building large-scale functional brain models," *Frontiers Neuroinform.*, vol. 7, p. 48, Jan. 2014.

[21] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers Neurosci.*, vol. 11, p. 682, Dec. 2017.

[22] S. B. Shrestha and G. Orchard, "SLAYER: Spike layer error reassignment in time," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2018, pp. 1412–1421.

[23] E. Falotico *et al.*, "Connecting artificial brains to robots in a comprehensive simulation framework: The neurorobotics platform," *Frontiers Neurorobotics*, vol. 11, p. 2, Jan. 2017.

[24] D. Rasmussen, "NengoDL: Combining deep learning and neuromorphic modelling methods," *Neuroinformatics*, vol. 17, no. 4, pp. 611–628, Oct. 2019.

[25] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: VGG and residual architectures," *Frontiers Neurosci.*, vol. 13, p. 95, Mar. 2019.

[26] J. Wu, Y. Chua, M. Zhang, G. Li, H. Li, and K. C. Tan, "A tandem learning rule for effective training and rapid inference of deep spiking neural networks," Jun. 2019, *arXiv:1907.01167*. [Online]. Available: http://arxiv.org/abs/1907.01167

[27] B. Rueckauer, C. Bybee, R. Goettsche, Y. Singh, J. Mishra, and A. Wild, "NxTF: An API and compiler for deep spiking neural networks on Intel Loihi," 2021, *arXiv:2101.04261*. [Online]. Available: https://arxiv.org/abs/2101.04261

[28] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith, "Benchmarking keyword spotting efficiency on neuromorphic hardware," in *Proc. 7th Annu. Neuro-Inspired Comput. Elements Workshop (NICE)*, 2019, pp. 1–8.

[29] T.-Y. Liu, A. Mahjoubfar, D. Prusinski, and L. Stevens, "Neuromorphic computing for content-based image retrieval," 2020, *arXiv:2008.01380*. [Online]. Available: http://arxiv.org/abs/2008.01380

[30] K. P. Patel, E. Hunsberger, S. Batir, and C. Eliasmith, "A spiking neural network for image segmentation," unpublished.

[31] R. Massa, A. Marchisio, M. Martina, and M. Shafique, "An efficient spiking neural network for recognizing gestures with a DVS camera on the Loihi neuromorphic processor," May 2020, *arXiv:2006.09985*. [Online]. Available: http://arxiv.org/abs/2006.09985

[32] N. Rathi, G. Srinivasan, P. Panda, and K. Roy, "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," May 2020, *arXiv:2005.01807*. [Online]. Available: http://arxiv.org/abs/2005.01807

[33] S. K. Esser *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proc. Nat. Acad. Sci. USA*, vol. 113, no. 41, pp. 11441–11446, Oct. 2016.

[34] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks," *IEEE Signal Process. Mag.*, vol. 36, pp. 61–63, 2019.

[35] F. Zenke and T. P. Vogels, "The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks," *Neural Comput.*, pp. 1–27, Jan. 2021, doi: 10.1162/neco_a_01367.

[36] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Proc. Conf. Neural Inf. Process. Syst. (NIPS)*. Red Hook, NY, USA: Curran Associates, 2018, pp. 787–797.

[37] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers Neurosci.*, vol. 12, p. 331, May 2018.

[38] H. H. See *et al.*, "ST-MNIST—The spiking tactile MNIST neuromorphic dataset," 2020, *arXiv:2005.04319*. [Online]. Available: http://arxiv.org/abs/2005.04319

[39] E. Ceolini *et al.*, "Hand-gesture recognition based on EMG and event-based camera sensor fusion: A benchmark in neuromorphic computing," *Frontiers Neurosci.*, vol. 14, p. 637, Aug. 2020.

[40] T. Taunyazov *et al.*, "Event-driven visual-tactile sensing and learning for robots," *Robot., Sci. Syst.*, vol. 4, p. 5, Jul. 2020.

[41] G. Tang, N. Kumar, and K. P. Michmizos, "Reinforcement co-learning of deep and spiking neural networks for energy-efficient mapless navigation with neuromorphic hardware," Mar. 2020, *arXiv:2003.01157*. [Online]. Available: http://arxiv.org/abs/2003.01157

[42] J. Weston *et al.*, "Towards AI-complete question answering: A set of prerequisite toy tasks," 2015, *arXiv:1502.05698*. [Online]. Available: http://arxiv.org/abs/1502.05698

[43] A. Santoro *et al.*, "A simple neural network module for relational reasoning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4967–4976.

[44] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," 2019, *arXiv:1906.02243*. [Online]. Available: http://arxiv.org/abs/1906.02243

[45] M. Jaderberg *et al.*, "Decoupled neural interfaces using synthetic gradients," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2017, pp. 1627–1635.

[46] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature Commun.*, vol. 7, no. 1, pp. 1–10, Dec. 2016.

[47] G. Bellec *et al.*, "A solution to the learning dilemma for recurrent networks of spiking neurons," *Nature Commun.*, vol. 11, no. 1, p. 3625, Dec. 2020.

[48] P. Baldi, P. Sadowski, and Z. Lu, "Learning in the machine: Random backpropagation and the deep learning channel," *Artif. Intell.*, vol. 260, pp. 1–35, Jul. 2018.

[49] K. Stewart, G. Orchard, S. B. Shrestha, and E. Neftci, "Online few-shot gesture learning on a neuromorphic processor," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 10, no. 4, pp. 512–521, Dec. 2020.

[50] T. DeWolf, P. Jaworski, and C. Eliasmith, "Nengo and low-power AI hardware for robust, embedded neurorobotics," Jul. 2020, *arXiv:2007.10227*. [Online]. Available: http://arxiv.org/abs/2007.10227

[51] P. T. P. Tang, T.-H. Lin, and M. Davies, "Sparse coding by spiking neural networks: Convergence theory and computational results," 2017, *arXiv:1705.05475*. [Online]. Available: http://arxiv.org/abs/1705.05475

[52] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen, "Sparse coding via thresholding and local competition in neural circuits," *Neural Comput.*, vol. 20, no. 10, pp. 2526–2563, Oct. 2008.

[53] S. Shapero, C. Rozell, and P. Hasler, "Configurable hardware integrate and fire neurons for sparse approximation," *Neural Netw.*, vol. 45, pp. 134–143, Sep. 2013.

[54] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Imag. Sci.*, vol. 2, no. 1, pp. 183–202, Jan. 2009.

[55] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar, "Cats and dogs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, Jun. 2012, pp. 3498–3505.

[56] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," *Ann. Statist.*, vol. 32, no. 2, pp. 407–499, 2004.

[57] M. Davies. *Exploring Neuromorphic Computing for AI: Why Spikes? (Part Two)*. [Online]. Available: https://www.intel.com/content/www/us/en/artificial-intelligence/posts/exploring-neuromorphic-computing-for-ai-why-spikes-part-two.html

[58] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, 2010, pp. 399–406.

[59] E. Kim, J. Yarnall, P. Shah, and G. T. Kenyon, "A neuromorphic sparse coding defense to adversarial images," in *Proc. Int. Conf. Neuromorphic Syst.*, Jul. 2019, pp. 1–8.

[60] C. Pehlevan, "A spiking neural network with local learning rules derived from nonnegative similarity matching," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 7958–7962.

[61] T.-H. Lin and P. T. P. Tang, "Dictionary learning by dynamical neural networks," May 2018, *arXiv:1805.08952*. [Online]. Available:

http://arxiv.org/abs/1805.08952
[62] Y. Watkins, A. Thresher, P. F. Schultz, A. Wild, A. Sornborger, and G. T. Kenyon, "Unsupervised dictionary learning via a spiking locally competitive algorithm," in *Proc. Int. Conf. Neuromorphic Syst.*, Jul. 2019, pp. 1–5.
[63] G. Schöner and J. Spencer, Eds., *Dynamic Thinking: A Primer on Dynamic Field Theory*. Oxford, U.K.: Oxford Univ. Press, 2015.
[64] Y. Sandamirskaya, "Dynamic neural fields as a step toward cognitive neuromorphic architectures," *Frontiers Neurosci.*, vol. 7, pp. 276–289, Jan. 2014.
[65] E. Neftci, J. Binas, U. Rutishauser, E. Chicca, G. Indiveri, and R. J. Douglas, "Synthesizing cognition in neuromorphic electronic systems," *Proc. Nat. Acad. Sci. USA*, vol. 110, no. 37, pp. E3468–E3476, Sep. 2013.
[66] Y. Sandamirskaya, "The importance of space and time for signal processing in neuromorphic agents," *IEEE Signal Process. Mag.*, vol. 36, no. 6, pp. 16–28, Dec. 2019.
[67] O. Lomp, M. Richter, S. K. U. Zibner, and G. Schöner, "Developing dynamic field theory architectures for embodied cognitive systems with cedar," *Frontiers Neurorobotics*, vol. 10, pp. 1–18, Nov. 2016.
[68] A. Renner, M. Evanusa, and Y. Sandamirskaya, "Event-based attention and tracking on neuromorphic hardware," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2019, pp. 1709–1716.
[69] M. B. Milde *et al.*, "Obstacle avoidance and target acquisition for robot navigation using a mixed signal analog/digital neuromorphic processing system," *Frontiers Neurorobotics*, vol. 11, pp. 11–28, Jul. 2017.
[70] S. J. Verzi *et al.*, "Computing with spikes: The advantage of fine-grained timing," *Neural Comput.*, vol. 30, no. 10, pp. 2660–2690, Oct. 2018.
[71] F. Ponulak and J. J. Hopfield, "Rapid, parallel path planning by propagating wavefronts of spiking neural activity," *Frontiers Comput. Neurosci.*, vol. 7, p. 98, Jul. 2013.
[72] K. E. Hamilton, T. M. Mintz, and C. D. Schuman, "Spike-based primitives for graph algorithms," 2019, *arXiv:1903.10574*. [Online]. Available: https://arxiv.org/abs/1903.10574
[73] K. Hamilton, T. Mintz, P. Date, and C. D. Schuman, "Spike-based graph centrality measures," in *Proc. Int. Conf. Neuromorphic Syst. (ICONS)*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1–8.
[74] A. Ali and J. Kwisthout, "A spiking neural algorithm for the network flow problem," 2019, *arXiv:1911.13097*. [Online]. Available: https://arxiv.org/abs/1911.13097
[75] L. Buesing, J. Bill, B. Nessler, and W. Maass, "Neural dynamics as sampling: A model for stochastic computation in recurrent networks of spiking neurons," *PLoS Comput. Biol.*, vol. 7, no. 11, Nov. 2011, Art. no. e1002211.
[76] C. Yakopcic, N. Rahman, T. Atahary, T. M. Taha, and S. Douglass, "Solving constraint satisfaction problems using the loihi spiking neuromorphic processor," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 1079–1084.
[77] J. B. Aimone, O. Parekh, C. A. Phillips, A. Pinar, W. Severa, and H. Xu, "Dynamic programming with spiking neural computing," in *Proc. Int. Conf. Neuromorphic Syst. (ICONS)*. New York, NY, USA: Association for Computing Machinery, Jul. 2019, pp. 1–9.
[78] S. M. Mniszewski, "Graph partitioning as quadratic unconstrained binary optimization (QUBO) on spiking neuromorphic hardware," in *Proc. Int. Conf. Neuromorphic Syst. (ICONS)*. New York, NY, USA: Association for Computing Machinery, Jul. 2019, pp. 1–5.
[79] M. Z. Alom, B. Van Essen, A. T. Moody, D. P. Widemann, and T. M. Taha, "Quadratic unconstrained binary optimization (QUBO) on neuromorphic computing system," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 3922–3929.
[80] X. Lagorce and R. Benosman, "STICK: Spike time interval computational kernel, a framework for general purpose computation using neurons, precise timing, delays, and synchrony," *Neural Comput.*, vol. 27, no. 11, pp. 2261–2317, Nov. 2015.
[81] X. Wang, T. Song, F. Gong, and P. Zheng, "On the computational power of spiking neural P systems with self-organization," *Sci. Rep.*, vol. 6, no. 1, pp. 1–16, Sep. 2016.
[82] G. Dragoi and S. Tonegawa, "Distinct preplay of multiple novel spatial experiences in the rat," *Proc. Nat. Acad. Sci. USA*, vol. 110, no. 22, pp. 9100–9105, May 2013.
[83] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998.
[84] R. K. Ahuja, K. Mehlhorn, J. Orlin, and R. E. Tarjan, "Faster algorithms for the shortest path problem," *J. ACM*, vol. 37, no. 2, pp. 213–223, Apr. 1990.
[85] M. Thorup, "On ram priority queues," in *Proc. 7th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1996, pp. 59–67.
[86] J. Hopfield and D. Tank, "Computing with neural circuits: A model," *Science*, vol. 233, no. 4764, pp. 625–633, Aug. 1986.
[87] Z. Jonke, S. Habenschuss, and W. Maass, "Solving constraint satisfaction problems with networks of spiking neurons," *Frontiers Neurosci.*, vol. 10, p. 118, Mar. 2016.
[88] W. Maass, "To spike or not to spike: That is the question," *Proc. IEEE*, vol. 103, no. 12, pp. 2219–2224, Dec. 2015.
[89] J. Binas, G. Indiveri, and M. Pfeiffer, "Spiking analog VLSI neuron assemblies as constraint satisfaction problem solvers," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 2094–2097.
[90] H. Mostafa, L. K. Müller, and G. Indiveri, "An event-based architecture for solving constraint satisfaction problems," *Nature Commun.*, vol. 6, no. 1, p. 8941, Dec. 2015.
[91] G. A. F. Guerra and S. B. Furber, "Using stochastic spiking neural networks on SpiNNaker to solve constraint satisfaction problems," *Frontiers Neurosci.*, vol. 11, p. 714, Dec. 2017.
[92] C. Ostrau, C. Klarhorst, M. Thies, and U. Rückert, "Comparing neuromorphic systems by solving Sudoku problems," in *Proc. Int. Conf. High Perform. Comput. Simul. (HPCS)*, Jul. 2019, pp. 521–527.
[93] A. Kugele and K. Meier, "Solving the constraint satisfaction problem Sudoku on neuromorphic hardware," M.S. thesis, Dept. Phys. Astron., Univ. Heidelberg, Heidelberg, Germany, 2018.
[94] J. Steidel, "Solving map coloring problems on analog neuromorphic hardware," M.S. thesis, Dept. Phys. Astron., Univ. Heidelberg, Heidelberg, Germany, 2018.
[95] D. Liang and G. Indiveri, "A neuromorphic computational primitive for robust context-dependent decision making and context-dependent stochastic computation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 66, no. 5, pp. 843–847, May 2019.
[96] G. Pedretti *et al.*, "A spiking recurrent neural network with phase-change memory neurons and synapses for the accelerated solution of constraint satisfaction problems," *IEEE J. Explor. Solid-State Comput. Devices Circuits*, vol. 6, no. 1, pp. 89–97, Jun. 2020.
[97] J. L. Gearhart, K. L. Adair, R. J. Detry, J. D. Durfee, K. A. Jones, and N. Martin, "Comparison of open-source linear programming solvers," Sandia Nat. Laboratories, Albuquerque, NM, USA, Tech. Rep. SAND2013-8847, 2013.
[98] C. Yakopcic, N. Rahman, T. Atahary, T. M. Taha, A. Beigh, and S. Douglass, "High speed cognitive domain ontologies for asset allocation using loihi spiking neurons," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–8.
[99] H. Jang, O. Simeone, B. Gardner, and A. Gruning, "An introduction to probabilistic spiking neural networks: Probabilistic models, learning rules, and applications," *IEEE Signal Process. Mag.*, vol. 36, no. 6, pp. 64–77, Nov. 2019.
[100] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck, "Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output," *Proc. IEEE*, vol. 102, no. 10, pp. 1470–1484, Oct. 2014.
[101] G. Gallego *et al.*, "Event-based vision: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Jul. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9138762
[102] K. Stewart, G. Orchard, S. B. Shrestha, and E. Neftci, "On-chip few-shot learning with surrogate gradient descent on a neuromorphic processor," in *Proc. 2nd IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Aug. 2020, pp. 223–227.
[103] N. J. Strausfeld and J. G. Hildebrand, "Olfactory systems: Common design, uncommon origins?" *Current Opinion Neurobiol.*, vol. 9, no. 5, pp. 634–639, Oct. 1999.
[104] N. Imam and T. A. Cleland, "Rapid online learning and robust recall in a neuromorphic olfactory circuit," *Nature Mach. Intell.*, vol. 2, no. 3, pp. 181–191, Mar. 2020.
[105] A. Vergara, J. Fonollosa, J. Mahiques, M. Trincavelli, N. Rulkov, and R. Huerta, "On the performance of gas sensor arrays in open sampling systems using inhibitory support vector machines," *Sens. Actuators B, Chem.*, vol. 185, pp. 462–477, Aug. 2013.
[106] T. DeWolf, T. C. Stewart, J.-J. Slotine, and C. Eliasmith, "A spiking neural model of adaptive arm control," *Proc. Roy. Soc. B, Biol. Sci.*, vol. 283, no. 1843, Nov. 2016, Art. no. 20162134.
[107] S. Glatz, J. Martel, R. Kreiser, N. Qiao, and Y. Sandamirskaya, "Adaptive motor control and learning in a spiking neural network realised on a mixed-signal neuromorphic processor," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 9631–9637.
[108] R. K. Stagsted, A. Vitale, A. Renner, L. B. Larsen, A. L. Christensen, and Y. Sandamirskaya, "Event-based PID controller fully realized in neuromorphic hardware: A one DoF study," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 10939–10944.
[109] R. S. Dimitrova, M. Gehrig, D. Brescianini, and D. Scaramuzza, "Towards low-latency high-bandwidth control of quadrotors using event cameras," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 4294–4300.
[110] A. Vitale, A. Renner, C. Nauer, D. Scaramuzza, and Y. Sandamirskaya, "Event-driven vision and control for UAVs on a neuromorphic chip," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Xi'an, China, May 2021.
[111] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: A review," *Neural Netw.*, vol. 21, no. 4, pp. 642–653, May 2008.
[112] S. Steingrube, M. Timme, F. Wörgötter, and P. Manoonpong, "Self-organized adaptation of a simple neural circuit enables complex robot behaviour," *Nature Phys.*, vol. 6, no. 3, pp. 224–230, Mar. 2010.
[113] I. Polykretis, G. Tang, and K. P. Michmizos, "An astrocyte-modulated neuromorphic central pattern generator for hexapod robot locomotion on Intel's Loihi," in *Proc. Int. Conf. Neuromorphic Syst.*, Jul. 2020, pp. 1–9.
[114] P. Balachandar and K. P. Michmizos, "A spiking neural network emulating the structure of the oculomotor system requires no learning to control a biomimetic robotic head," 2020, *arXiv:2002.07534*. [Online]. Available: http://arxiv.org/abs/2002.07534
[115] D. Ryczko, A. Simon, and A. J. Ijspeert, "Walking with salamanders: From molecules to biorobotics," *Trends Neurosciences*, vol. 43, no. 11, pp. 916–930, Nov. 2020.
[116] D. Gutierrez-Galan, J. P. Dominguez-Morales, F. Perez-Peña, A. Jimenez-Fernandez, and A. Linares-Barranco, "Neuropod: A real-time neuromorphic spiking CPG applied to robotics," *Neurocomputing*, vol. 381, pp. 10–19, Mar. 2020.
[117] N. Korkmaz, İ. Öztürk, and R. Kiliç, "Modeling, simulation, and implementation issues of CPGs for neuromorphic engineering applications," *Comput.*

Appl. Eng. Educ., vol. 26, no. 4, pp. 782–803, Jul. 2018.
[118] A. Spaeth, M. Tebyani, D. Haussler, and M. Teodorescu, "Neuromorphic closed-loop control of a flexible modular robot by a simulated spiking central pattern generator," in *Proc. 3rd IEEE Int. Conf. Soft Robot. (RoboSoft)*, May 2020, pp. 46–51.
[119] M. Folgheraiter, A. Keldibek, B. Aubakir, G. Gini, A. M. Franchi, and M. Bana, "A neuromorphic control architecture for a biped robot," *Robot. Auto. Syst.*, vol. 120, Oct. 2019, Art. no. 103244.
[120] M. J. Milford, G. F. Wyeth, and D. Prasser, "RatSLAM: A hippocampal model for simultaneous localization and mapping," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Apr. 2004, pp. 403–408.
[121] R. Kreiser, G. Waibel, N. Armengol, A. Renner, and Y. Sandamirskaya, "Error estimation and correction in a spiking neural network for map formation in neuromorphic hardware," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 6134–6140.
[122] R. Kreiser *et al.*, "An on-chip spiking neural network for estimation of the head pose of the iCub robot," *Frontiers Neurosci.*, vol. 14, p. 551, Jun. 2020, doi: 10.3389/fnins.2020.00551.
[123] G. Tang, A. Shah, and K. P. Michmizos, "Spiking neural network on neuromorphic hardware for energy-efficient unidimensional SLAM," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Nov. 2019, pp. 4176–4181.
[124] M. Hampo *et al.*, "Associative memory in spiking neural network form implemented on neuromorphic hardware," in *Proc. Int. Conf. Neuromorphic Syst.*, Jul. 2020, pp. 1–8.
[125] P. Farr, A. M. Jones, T. Bihl, J. Boubin, and A. DeMange, "Waveform design implemented on neuromorphic hardware," in *Proc. IEEE Int. Radar Conf. (RADAR)*, Apr. 2020, pp. 934–939.
[126] J. D. Smith *et al.*, "Solving a steady-state PDE using spiking networks and neuromorphic hardware," in *Proc. Int. Conf. Neuromorphic Syst.*, Jul. 2020, pp. 1–8.
[127] J. Wu, C. Xu, D. Zhou, H. Li, and K. C. Tan, "Progressive tandem learning for pattern recognition with deep spiking neural networks," Jul. 2020, *arXiv:2007.01204*. [Online]. Available: http://arxiv.org/abs/2007.01204
[128] M. Zhang, N. Zheng, D. Ma, G. Pan, and Z. Gu, "Efficient spiking neural networks with logarithmic temporal coding," 2018, *arXiv:1811.04233*. [Online]. Available: http://arxiv.org/abs/1811.04233
[129] A. Bagheri, O. Simeone, and B. Rajendran, "Training probabilistic spiking neural networks with first-to-spike decoding," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2018, pp. 2986–2990.
[130] H. Tanaka, D. Kunin, D. L. K. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," 2020, *arXiv:2006.05467*. [Online]. Available: http://arxiv.org/abs/2006.05467
[131] B. Scellier and Y. Bengio, "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation," *Frontiers Comput. Neurosci.*, vol. 11, p. 24, May 2017.
[132] T. J. Hamilton, C. Jin, J. Tapson, and A. van Schaik, "A 2-D cochlea with Hopf oscillators," in *Proc. IEEE Biomed. Circuits Syst. Conf.*, Nov. 2007, pp. 91–94.
[133] C. Warner and F. T. Sommer, "A model for image segmentation in retina," 2020, *arXiv:2005.02567*. [Online]. Available: https://arxiv.org/abs/2005.02567
[134] G. Pasquale, C. Ciliberto, F. Odone, L. Rosasco, and L. Natale, "Are we done with object recognition? The iCub robot's perspective," *Robot. Auto. Syst.*, vol. 112, pp. 260–281, Feb. 2019.
[135] S. D. Levy and R. Gayler, "Vector symbolic architectures: A new building material for artificial general intelligence," in *Proc. Conf. Artif. General Intell.*, 2008, pp. 414–418.
[136] E. P. Frady and F. T. Sommer, "Robust computation with rhythmic spike patterns," *Proc. Nat. Acad. Sci. USA*, vol. 116, no. 36, pp. 18050–18059, Sep. 2019.
[137] E. Paxon Frady, S. Kent, B. A. Olshausen, and F. T. Sommer, "Resonator networks for factoring distributed representations of data structures," 2020, *arXiv:2007.03748*. [Online]. Available: http://arxiv.org/abs/2007.03748
[138] W. Severa, C. M. Vineyard, R. Dellana, S. J. Verzi, and J. B. Aimone, "Training deep neural networks for binary communication with the whetstone method," *Nature Mach. Intell.*, vol. 1, no. 2, pp. 86–94, Feb. 2019.
[139] C. D. Schuman, J. P. Mitchell, R. M. Patton, T. E. Potok, and J. S. Plank, "Evolutionary optimization for neuromorphic systems," in *Proc. Neuro-Inspired Comput. Elements Workshop (NICE)*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1–9.
[140] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator," *eLife*, vol. 8, Aug. 2019, Art. no. e47314.
[141] H. Hazan *et al.*, "BindsNET: A machine learning-oriented spiking neural networks library in Python," *Frontiers Neuroinform.*, vol. 12, p. 89, Dec. 2018.
[142] Y. Pierre, "PyNN: A common interface for neuronal network simulators," *Frontiers Neuroinform.*, vol. 2, p. 11, Jan. 2008.
[143] J. B. Aimone, W. Severa, and C. M. Vineyard, "Composing neural algorithms with Fugu," in *Proc. Int. Conf. Neuromorphic Syst. (ICONS)*. New York, NY, USA: Association for Computing Machinery, Jul. 2019, pp. 1–8.
[144] C. Eliasmith *et al.*, "A large-scale model of the functioning brain," *Science*, vol. 338, no. 6111, pp. 1202–1205, Nov. 2012.

ABOUT THE AUTHORS

Mike Davies received the B.S. and M.S. degrees from the California Institute of Technology (Caltech), Pasadena, CA, USA, in 1998 and 2000, respectively.

He was a Founding Employee of Fulcrum Microsystems, Calabasas, CA, USA, and the Director of its Silicon Engineering Group until Intel's acquisition of Fulcrum in 2011. He led the development of four generations of low latency, highly integrated Ethernet switches using Fulcrum's proprietary asynchronous design methodology. Since 2014, he has been researching neuromorphic circuits, architectures, and algorithms at Intel Labs, Intel Corporation, Santa Clara, CA, USA. He is currently a Senior Principal Engineer and the Director of the Neuromorphic Computing Lab, Intel Corporation.

Andreas Wild received the Dr.Rer.Nat degree in physics with a focus on the development of silicon-based electron spin qubits from the Technical University of Munich, Munich, Germany, in 2013.

Since 2015, he has been a Senior Researcher with the Neuromorphic Computing Lab, Intel Corporation, Santa Clara, CA, USA, leading architecture and algorithms modeling.

Garrick Orchard received the Ph.D. degree from Johns Hopkins University, Baltimore, MD, USA, in 2012.

He joined the newly formed Singapore Institute for Neurotechnology (SINAPSE), National University of Singapore, Singapore. In 2019, he joined the Neuromorphic Computing Lab, Intel Corporation, Santa Clara, CA, USA, as a Senior Researcher focusing on sensing and perception.

Dr. Orchard was awarded the Temasek Research Fellowship from the Singapore Ministry of Defence in 2015.

Yulia Sandamirskaya received the Dr.Rer.Nat degree in physics with a focus on dynamic neural fields from Ruhr-Universität Bochum, Bochum, Germany, in 2010.

She led the Neuromorphic Cognitive Robots Group, Institute of Neuroinformatics (INI), University of Zurich, Zürich, Switzerland, and ETH Zürich, Zürich. She is currently a Senior Researcher with the Neuromorphic Computing Lab, Intel Corporation, Santa Clara, CA, USA, focusing on applications research.

Gabriel A. Fonseca Guerra received the Ph.D. degree in computer science with a focus on stochastic processes for neuromorphic hardware from The University of Manchester, Manchester, U.K., in 2020.

He is currently a Research Scientist with Neuromorphic Computing Lab, Intel Corporation, Santa Clara, CA, USA, working on algorithms and architecture development.

Prasad Joshi received the M.S. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 2008.

He is currently a Senior Research Scientist with Intel Labs, Intel Corporation, Santa Clara, CA, USA, where he manages a Silicon Research Team, Neuromorphic Computing Lab. He led the physical implementation of Loihi. His research interests include asynchronous design methods and design automation, as well as algorithms characterization on neuromorphic architecture.

Philipp Plank received the M.S. degree in biomedical engineering from the Graz University of Technology, Graz, Austria, in 2019.

He is currently a Research Scientist with Intel Labs, Intel Corporation, Santa Clara, CA, USA, working on algorithms and architecture modeling as a member of the Neuromorphic Computing Lab.

Sumedh R. Risbud received the Ph.D. degree in chemical and biomolecular engineering with a focus on theoretical fluid mechanics and microfluidics from Johns Hopkins University, Baltimore, MD, USA, in 2013.

He is currently a Research Scientist with Intel Labs, Intel Corporation, Santa Clara, CA, USA. His focus at the Neuromorphic Computing Lab is on algorithms and architecture development.