Darwin-S: A Reference Software Architecture for Brain-Inspired Computers

Shuiguang Deng, Pan Lv, and Ouwen Jin, Zhejiang University
Schahram Dustdar, Distributed Systems Group
Ying Li, De Ma, Zhaohui Wu, and Gang Pan, Zhejiang University

With the reduction of the semiconductor process size, the problems of "memory wall" and "power wall" in von Neumann architectures are becoming increasingly prominent. To solve these problems, brain-inspired computing unifies computing and storage.

With the development of the semiconductor industry, the process size of chips has reached the nanometer level. The problems of "storage wall" and "power wall" inherent in von Neumann architectures are becoming increasingly noticeable. As an abstract computing system, the human brain has very high parallel computing capability (simultaneously performing visual recognition, reasoning, body control, and movement) but low energy consumption (about 20 W). There are two striking differences between the computing principles of human brains and von Neumann computers. First, the human brain has a network of billions of neurons interconnected through trillions of synapses. Neurons are the computational primitive elements of the brain that exchange or transfer information through discrete action potentials or spikes, and synapses are the storage elements underlying memory and learning. The human brain unifies computing and storage through neurons and synapses, solving memory wall problems. Second, neurons in the brain transfer information through event-driven or time-dependent spikes. Spike-based temporal processing allows sparse and efficient information transfers in the brain, resulting in impressive low power consumption, which solves the problems of power wall.[1]

A spiking neural network (SNN), as shown in Figure 1 and categorized as the third generation of neural networks, is the most representative concept in the field of brain-inspired computing. Imitating the brain in terms of both neuron and synaptic connection models, SNNs feature rich spatial-temporal information and event-driven and high biological plausibility.

The advantage of low power consumption of brain-inspired computing cannot be fully exploited by running an SNN on a general-purpose processor. Researchers have launched several projects to develop neuromorphic chips and computers, including BrainScaleS,[2] SpiNNaker,[3] TrueNorth,[4] Loihi,[5] Tianjic,[6] and Darwin.[7]

Digital Object Identifier 10.1109/MC.2022.3144397
Date of current version: 6 May 2022

Efficiently managing and utilizing brain-inspired computer hardware resources has remained an unsolved challenge due to their large scale and complexity. BrainScaleS' operating system (OS),[8] developed by Heidelberg University, creates software development and a runtime environment for the BrainScaleS system. The software system initiated by the University of Manchester in the United Kingdom for the SpiNNaker system[9] comprises software programs executing on the SpiNNaker system and the software toolchain SpiNNTools.[10] IBM established a relatively complete software ecosystem around its TrueNorth chip, covering system, software, application, and other aspects.[11] Intel's Loihi development toolchain designs a Python-based application programming interface, compiler, runtime, a software simulator, and field-programmable gate array (FPGA) emulator.

The main software-related work of various brain-inspired computing systems focuses on training, model conversion, mapping, emulators, and so forth. Meanwhile, all software systems' support in runtime environments is merely considered from the perspective of a runtime library or a traditional OS (such as TrueNorth adopting a Linux system and SpiNNaker adopting a real-time OS). Therefore, we propose a more complete and concise software reference architecture called *Darwin-S*. It involves a brain-inspired OS and an integrated development environment (IDE).

A REFERENCE SOFTWARE ARCHITECTURE FOR BRAIN-INSPIRED COMPUTERS

Darwin-S aims to provide application development and operation reference architecture for brain-inspired computers so that researchers can develop applications without having to understand implementation details of the underlying hardware. The application of brain-inspired computers mainly includes a spike codec process and an SNN. The Spike codec process encodes input data (such as images) into the spike sequence and decodes the output spike sequence. The SNN processes spikes through its internal multilayer neurons and sends out result spikes through the output layer. Darwin-S entails an IDE that supports application development and an OS that provides a running environment for applications.

The idea of hierarchy and modularization can well realize decoupling between software architectures. Right-level abstraction makes the implementation details of hierarchy transparent to each other, which brings favorable extensibility and compatibility, as shown in Figure 2.

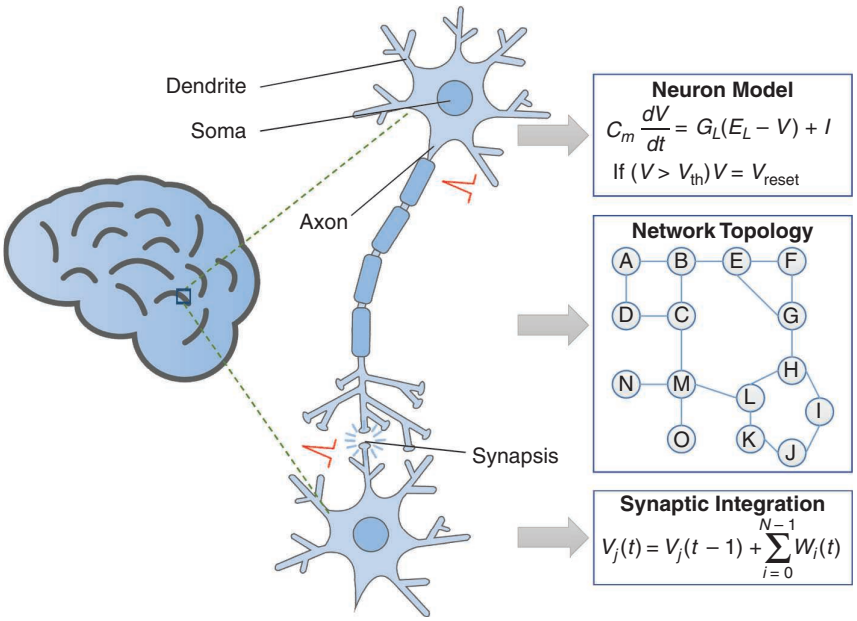

FIGURE 1. The structure of an SNN.

A general-purpose architecture

Darwin-S is designed to be applicable across different underlying hardware architectures, and this generality is fundamentally based on our abstraction of brain-inspired computers. The architecture and implementation methods among different brain-inspired hardware platforms are utterly different. However, the basic design principle is the same:

- using a large number of specially designed neurosynaptic cores (for example, crossbars on Loihi and TrueNorth, and ARM cores on SpiNNaker)
- simulating neurons dynamics in parallel
- storing synapses' weight data within or near each neurosynaptic core

- neurosynaptic cores switch spiking messages through specially designed connection paths (for example, networks on chip or Ethernet).

Based on these common features of brain-inspired computers, we have made two critical attempts to achieve hardware decoupling as follows.

First, we adopted a hardware abstract layer in a brain-inspired OS to shield the hardware differences of brain-inspired computers and abstract the hardware into three parts: storage, communication, and neural resources. Second, an intermediate model definition language to describe the upper SNN is involved in the brain-inspired application IDE, which makes the lower implementation entirely transparent for the upper layer development environment.

Brain-inspired OS

Compared with the concept of task and resource in traditional computer system software, we defined the SNN running on a brain-inspired computer as a *neural task*. Accordingly, neurons and synapses are termed as *neural resources*. Therefore, the brain-inspired OS is a customized software system for neural task operation and neural resource management.

From a hardware perspective, it implements the encapsulation and shielding of underlying neuromorphological computing resources, abstracts them as neural resources for unified management and scheduling, and supports operation of the upper

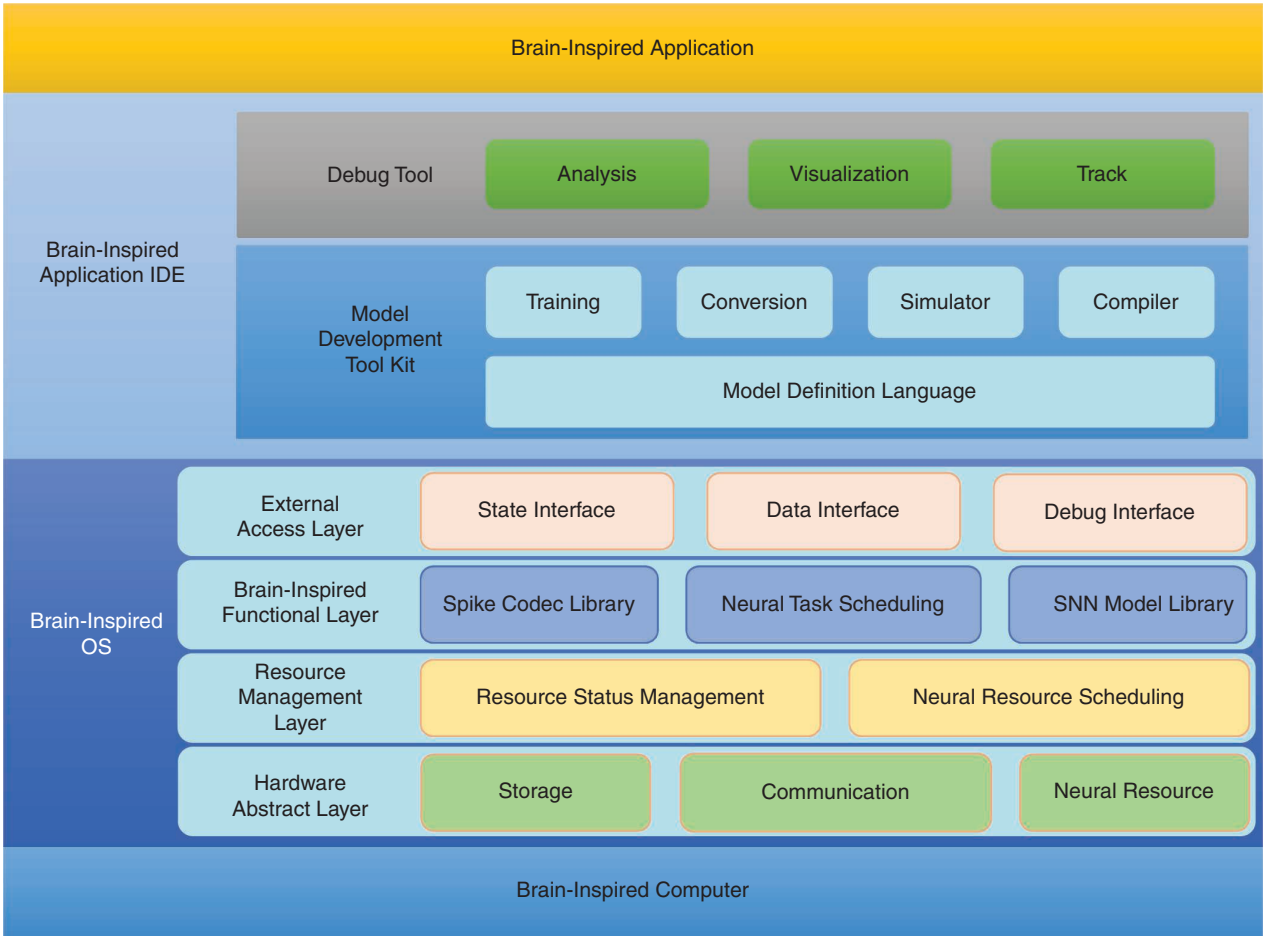

FIGURE 2. The reference software architecture for a brain-inspired computer.

neural tasks. From a software perspective, the OS dynamically allocates neural resources for neural tasks and provides an operating environment for scheduling multiple neural tasks. Concurrently, it also provides the internal information interface of neural resources and a control spike release process, and offers debugging means for users to develop neural tasks. The OS consists of the following four layers:

1. The hardware abstraction layer enables standardized hardware access interfaces to be compatible with different hardware.
2. The resource management layer realizes neural resource management and scheduling.
3. The brain-inspired functional layer supports the operation of neural tasks by providing a spike codec library, neural task scheduling, and an SNN model library.
4. The external access layer provides three kinds of access interfaces for the state, data, and debugging of the brain-inspired computer.

Brain-inspired application IDE

At present, the development of SNN applications is closely related to the hardware of brain-inspired computers. Therefore, we propose an application development and debugging tool, which enables users to efficiently build and debug an SNN application without caring about the underlying hardware.

> THE CRITICAL FUNCTIONS OF DARWINOS ARE THE MANAGEMENT AND SCHEDULING OF NEURAL RESOURCES AND TASKS.

The model development kit supports the use of direct training to obtain an SNN, or the conversion of a trained artificial neural network (ANN) to an SNN. We define a model description language to uniformly describe the SNNs obtained in these two methods. Then the SNN is simulated by a simulator. The model parameters are optimized according to the execution results to better performance. The compiler maps spiking neurons and synapses in the SNN to neurosynaptic cores. Finally, the SNN is compiled into an executable model code that the brain-inspired computer can run. After the application is deployed and running, users can observe the execution of the application through the visualization and analysis functions provided by the debugging tool.

DARWIN-S: THE IMPLEMENTATION

The specific implementation version of Darwin-S focuses on the Darwin application IDE (DarwinIDE) and Darwin brain-inspired OS (DarwinOS). DarwinIDE realizes the functions of application debugging and model development, ranging from a conversion tool, simulator, and compiler, to model definition language. DarwinOS can support the running of the application and management of the brain-inspired computer.

DarwinOS: Management and scheduling

DarwinOS serves as the primary software operation platform of the Darwin brain-inspired computer. It makes full use of the computer's distributed architecture based on a hybrid computing architecture. An ARM chip realized by an FPGA completes the logic part of the OS, and the neural task is fulfilled by a neuromorphic chip, the Darwin 2. The critical functions of DarwinOS are the management and scheduling of neural resources and tasks, as depicted in Figure 3.

Darwin Mouse: A Darwin brain-inspired computer. Darwin Mouse, a brain-inspired computer, is developed on the basis of neuromorphic chips, the Darwin 2. It adopts a mesh and tree hybrid architecture and supports 120 million neurons and 72 billion synapses. As displayed in Figure 3, on the first level, the Darwin 2 chip is organized in a mesh structure and connected by a high-speed, interchip interface. On the second level, the chips communicate through an FPGA, and on the third level the FPGA communicates with each other through Ethernet to form a tree structure. The mesh structure is adopted at the lowest level, which uses more local communication and less cross-regional communication of neuromorphic chips. considers the delay of cross-regional communication while ensuring the link throughput. However, the disadvantage of the

mesh structure is that it cannot connect a large number of neuromorphic chips, so the highest level adopts the tree structure to solve this problem. We take the second level as the basic module of Darwin Mouse. Darwin Mouse consists of 66 such modules, and each basic module contains 12 interconnected chips and an FPGA bridge. The latter is responsible for spike transmission between the underlying chips and interaction with the other basic module. The connected chip transmits the spike within the chip or between adjacent chips through the on-chip network, as shown in Figure 4.

Hardware abstract layer. This layer shields the hardware differences of brain-inspired computers and achieves the hardware-related abstraction of storage, communication, and neural resources so that the reference architecture can sustain different brain-inspired hardware.

Drawing on the three-tier hybrid architecture of Darwin Mouse, we implement real-time communication middleware and a distributed file system to facilitate real-time communication between FPGAs and the storage of large-scale SNN model files. By means of a publish/subscribe mode, the real-time communication middleware featuring data ensures quality-of-service strategies and makes it possible to transmit spikes between FPGAs simultaneously, efficiently, and flexibly. The distributed file system uniformly manages and abstracts all the storage resources of Darwin Mouse and

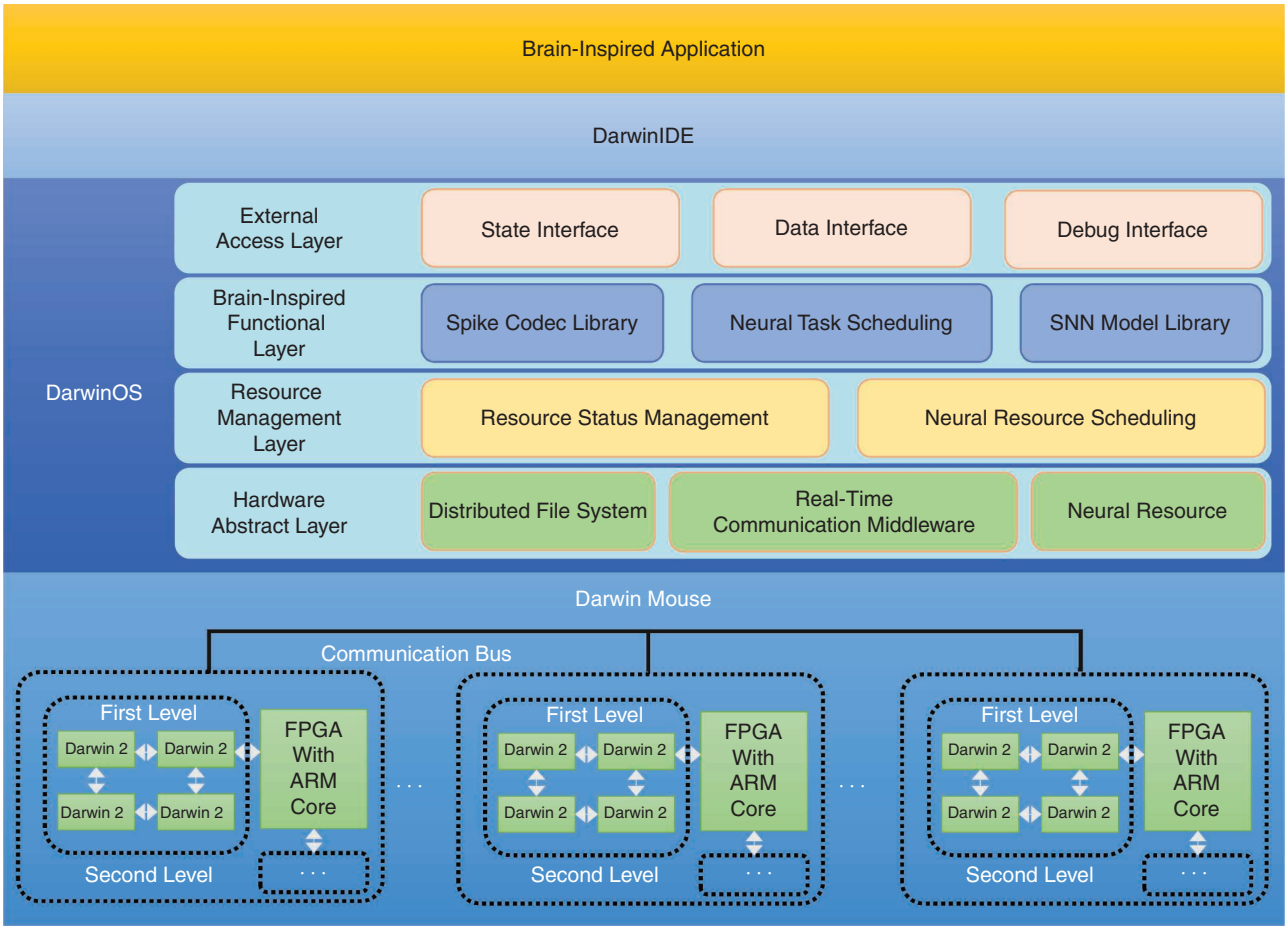

FIGURE 3. The implementation of a brain-inspired OS for a Darwin brain-inspired computer.

creates a unified file operation interface for the upper software. At the same time, the distributed file system supports redundant backup, which improves the reliability of file storage. Neural resource abstraction unifies the operation of neurons, synapses, and other neural resources in neuromorphic chips into a standard operation interface. It mainly includes neuron behavior configuration, synaptic weight parameter read/write, neuron connection routing configuration, and so on. This design makes it possible to support a wide variety of types of neuromorphic chips.

Resource management layer. DarwinOS realizes the state management and scheduling of neural resources through the resource management layer. Its resource management ability and allocation efficiency directly determine the overall performance of Darwin Mouse.

The primary resource state management is composed of the occupation of neuron resources and synaptic resources, neuron membrane potential, and failure state. On the one hand, resource state management can offer external users overall resource occupancy and system failure state. On the other hand, it can set resource constraints for neural resource scheduling. Neural resource scheduling performs dynamic neural task migration aligned with the resource requirements of different neural tasks and the current resource's idle state. Through resource scheduling, the neural tasks that lack a long-time input spike are primarily exported to the external file system. The neural resources are subsequently allocated to the neural tasks that need to run urgently to meet the computing needs of users' neural tasks to the greatest extent. The system's optimal power consumption and performance can be fully achieved by reasonable allocation of neural resources through load balancing.

Brain-inspired functional layer. The brain-inspired functional layer provides a runtime environment for neural tasks, such as spike encoding and decoding, neural task scheduling, and the SNN model library. It enables

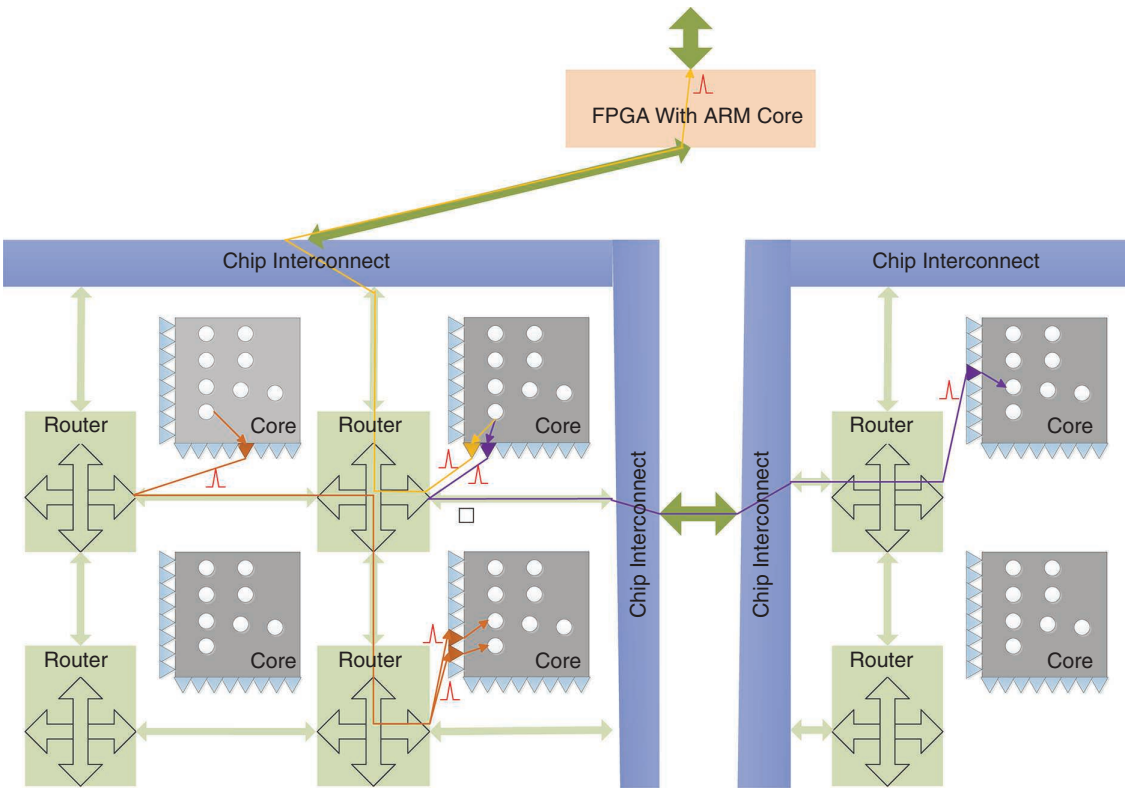

FIGURE 4. The basic module structure of Darwin Mouse.

the brain-inspired computer to support large-scale neural tasks such as brain simulation and the application of small-scale neural tasks in scenes like edge computing.

The information is transmitted and processed by discrete spikes within the SNN, so the input data of smell, image, and voice must be coded into a spike sequence. The brain-inspired OS provides a spike codec library for application. The brain-inspired computer can run neural tasks with the assistance of neural task scheduling, which consists of neural task decomposition, neural resource allocation, dynamic network mapping, and neural network loading. Based on the occupation of neural resources, the neural task, disassembled into multiple executive subtasks, is dynamically mapped and loaded into the neuromorphic chip of a brain-inspired computer for operation.

The SNN model library contains trained standard models, such as voice and image recognition, and has a corresponding spike codec library. The OS can dynamically load different SNNs and codec libraries to realize different brain-inspired applications in accordance with user needs.

External access layer. To tackle the external interaction problem of brain-inspired computers, the OS mainly provides the neural resource state access interface and fulfills input and output operations of the SNN and its spike encoding and decoding file through the data interaction interface. Users can call the debugging interface to obtain the running state of the application, especially neurosynaptic core information such as membrane voltage.

DarwinIDE: Development, language, and compiler. When neuroscientists or SNN algorithm researchers develop brain-inspired models based on a Darwin brain-inspired computer, they confront big challenges

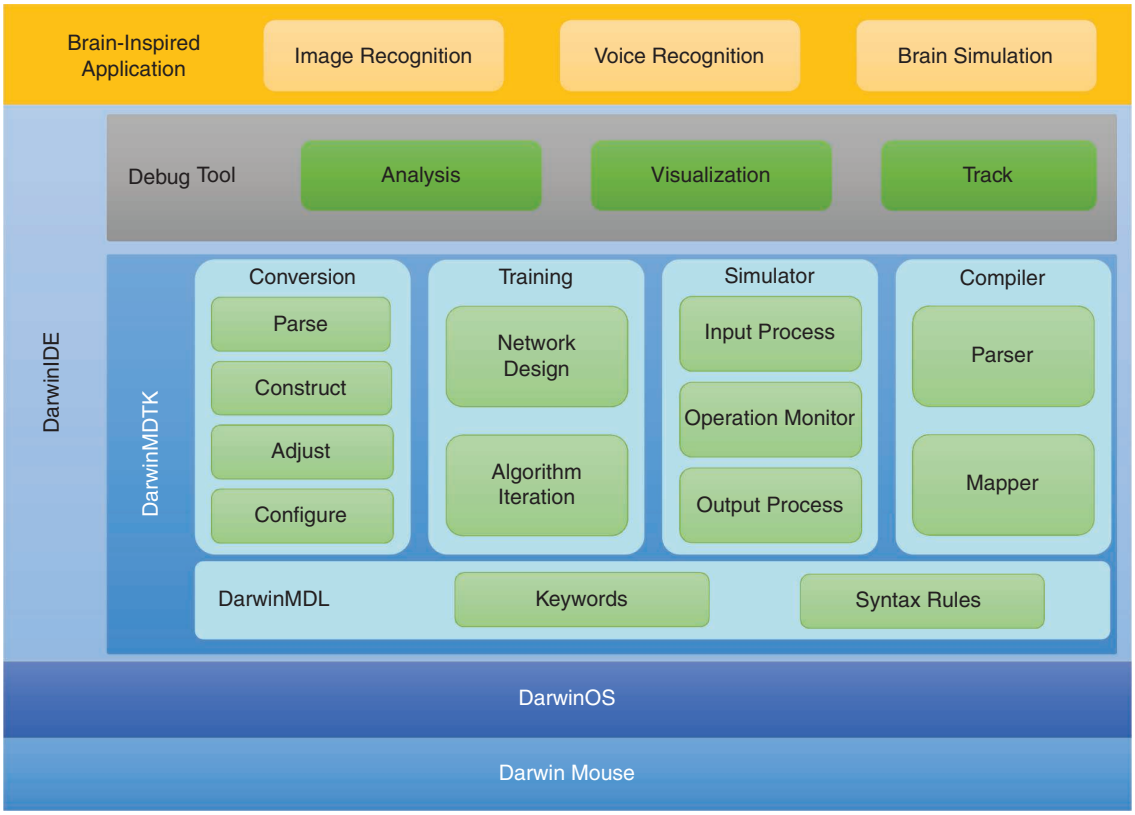

FIGURE 5. The implementation of a brain-inspired application IDE. DarwinMDTK: Darwin model development tool kit; DarwinMDL: Darwin model definition language.

due to an insufficient understanding of hardware constraints. Therefore, to enable users to efficiently build neural networks with resource constraints and observe the operation of neural networks, we implement DarwinIDE based on a Darwin-S software reference architecture. As presented in Figure 5, DarwinIDE consists of two parts: a Darwin model development tool kit (DarwinMDTK) and debugging tool. DarwinMDTK implements model training, conversion, simulation, and compilation. The debugging tool has the functions of visualization, analysis, and execution tracking.

Due to massive neurons in the neural network (up to billions of neurons), it is impossible to manually complete the neurons' connection and the configuration of weight parameters. Instead, the construction of the SNN needs to be completed automatically through DarwinMDTK. First, the SNN is trained directly by the training tool, or the ANN is converted into the SNN described by Darwin model definition language (DarwinMDL) using the conversion tool, and then the model is simulated by the simulator. After the simulation run is passed, the compiler parses the model, completes the model mapping, and converts it into a hardware-executable model code. The debugging tool is oriented to the execution process of the executable model. It obtains the information of neuron spike release and membrane voltage by calling the debugging interface provided by the OS and displays it graphically to analyze or track.

Model definition language. DarwinMDL specifies standardized description syntax rules and keywords with high neural correlation so as to describe the SNN more accurately and standardized. As a language independent of a hardware platform, DarwinMDL provides an intermediate description language for the SNN model using training, transformation, simulation, and compilation tools. The SNN described by DarwinMDL can also be displayed synchronously through graphics to visually check the correctness of the internal structure of the SNN, and to minimize editing and syntax errors.

THE TRAINING PROCESS OF SNNS IS SIMILAR TO THAT OF DEEP NEURAL NETWORKS, EXCEPT THAT SNNS NEED TO ENCODE INFORMATION WITH SPIKES.

Model conversion. At present, there are primarily two ways to achieve the SNN. One is to obtain the available SNN by adjusting and converting the trained ANN through the neural network layer and fine-tuning the connection's weight parameters.[12] Another approach is to obtain an SNN through a direct training algorithm.[13] These large-scale SNNs that yield state-of-the-art performance are most likely obtained by conversion-based approaches.[14,15] Considering this aspect, we include a model-conversion tool in our DarwinMDTK to help developers convert their ANN trained with PyTorch or TensorFlow into the SNN described by DarwinMDL. Moreover, a detailed instance will be presented in the "Case Study" section to clarify this workflow.

The workflow of the tool can be divided into four main steps. The first step is model parsing. After the deep learning library is used to build the ANN and the training is completed, the model parser parses the ANN to generate an intermediate representation of the model.[16] Some neural network layers in the ANN are removed or replaced according to the need, and an intermediate ANN is obtained. Next, an equivalent SNN model structure is constructed for the connection relationship between neurons in the ANN. Afterward, the weight parameters in the SNN are further adjusted and then assigned to the corresponding synapse of the SNN. Finally, the parameters of the neurons and synapses in the SNN are configured to accomplish the construction of the SNN.

Model training. In addition to conversion, direct training is another way to obtain an SNN. To enable developers to cultivate SNNs as efficiently as possible from scratch, we integrated an SNN training framework called *Spike-based Artificial Intelligence Computing (SPAIC)* (https://github.com/ZhejianglabNCRC/SPAIC) into DarwinMDTK. Developers need only define the SNN network structure, training algorithm, and relevant training parameters, and the training process will be automatically carried out. The output of this process is the SNN defined by DarwinMDL, which is ready to be used in the next step of the simulation.

Several training algorithms, such as the spatio-temporal credit assignment one,[17] are integrated into the SPAIC. The training process of SNNs is similar to that of deep neural networks, except that SNNs need to encode information with spikes. A typical training process is as follows. First, according to the design of the network structure, network connections (synapses) and parameters are initialized. Then, several iterations of the learning algorithm are carried out. Each iteration usually consists of a forward pass, where the SNN is simulated for certain time steps with the inputs (outputs) and encoded (decoded) to (from) the spikes, and a backward pass, where the error is backpropagated through spikes to the network parameters, and then the parameters are updated by an optimizer according to the gradient. Finally, the trained SNN will be packaged into a DarwinMDL standard format.

Model simulator. Before compiling the application, the simulator is used to simulate the converted SNN to evaluate the model performance and decide whether to adjust the conversion parameters for optimization. The simulator is divided into three parts from top to bottom: the input process, operation monitoring of the SNN, and model output process. The input process provides the pretreatment and encoding of input data. The encoding data are fed into the SNN executing on the simulator. The operational status of the SNN, including neurons, synaptic connections, and firing spikes, will be continuously monitored during operation. Ultimately, the output of the model needs to be processed. If the frequency coding method is adopted, each neuron's spike excitation in the final output layer of the SNN needs to be counted to achieve the final result.

In executing the SNN on the simulator and processing data, the neuron state of each layer is monitored by defining the neuron state monitor, and the spike excitation of the neurons at each layer is obtained by defining the spike-excitation monitor. On the one hand, the two monitors can monitor neurons' state changes for the performance analysis of the model, and analyze and deal with neurons that fire too fast or do not fire for a long time. On the other hand, the performance bottleneck in the model-calculation process can be analyzed in time during the simulation application. The model can be iterated and optimized in time through simulation at the software level to obtain better performance when it is deployed on hardware.

Model compiler. The compiler consists of a language parser and a mapper. The language parser converts the model into an intermediate code, syntax tree for mapping. Then the mapper binds it to specific neural resources.

- *Model language parser*: The process of language parsing, like a traditional compiler, undergoes the stages of lexical, syntax, and semantic analyses. A lexical analysis scans and identifies words, compares keywords, and establishes corresponding symbol lists. A syntax analysis identifies corresponding syntax categories according to the syntax rules of DarwinMDL, while a semantic analysis checks the overall network's model structure. Additionally, error checking and handling are performed at each stage. In the semantic analysis stage, we optimize the compilation in terms of the characteristics of the neural network. During construction of the SNN, the order of the definitions of layers and the order of neurons' attributes in a single layer is utterly distinct. Additionally, some parameters can be omitted and default values can be used. Language analysis needs to optimize semantic parsing, obtain the correct hierarchical order according to the context, and put the value into the corresponding attribute key to complete semantic matching.
- *Model mapping*: The mapping of the SNN falls into two stages: coarse- and fine-grained distribution. The coarse-grained distribution stage aims to determine whether a brain-inspired computer can meet the demand of network mapping. If the resources of a brain-inspired computer can fulfill network requirements, the next fine-grained allocation phase will continue. After inputting the SNN to be mapped, a resource evaluation is executed. The total number of resources required by the network is calculated. If the results do not exceed the available resources of the hardware, the next calculation can be executed. Otherwise, the network cannot be mapped to the hardware. According to different SNN structures, a fine-grained allocation stage can be accomplished in two ways. The number of neurons allocated to each node can be calculated if the SNN is topographically well structured, such as a fully connected layer, convolutional

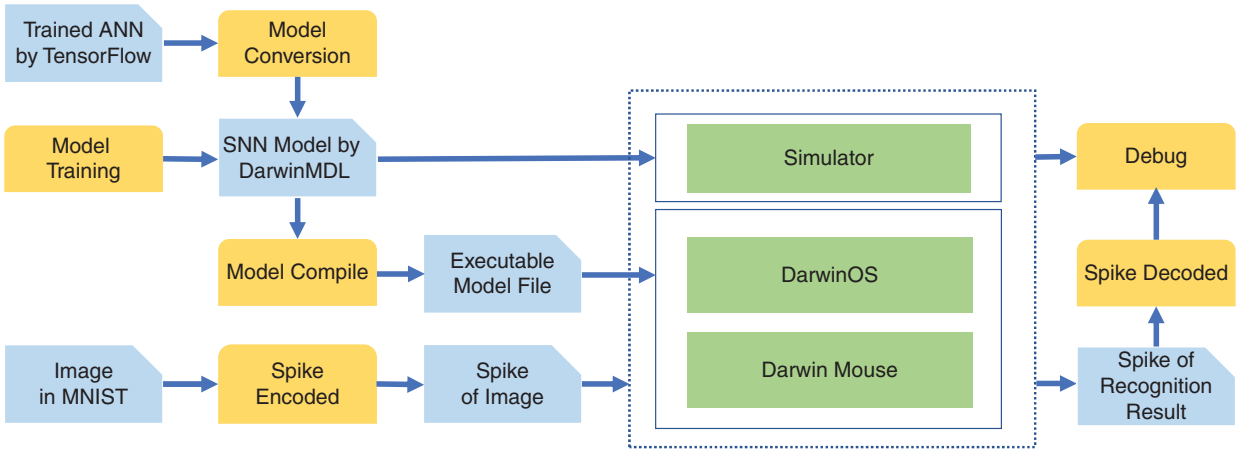

FIGURE 6. SNN model development, and the execution process.

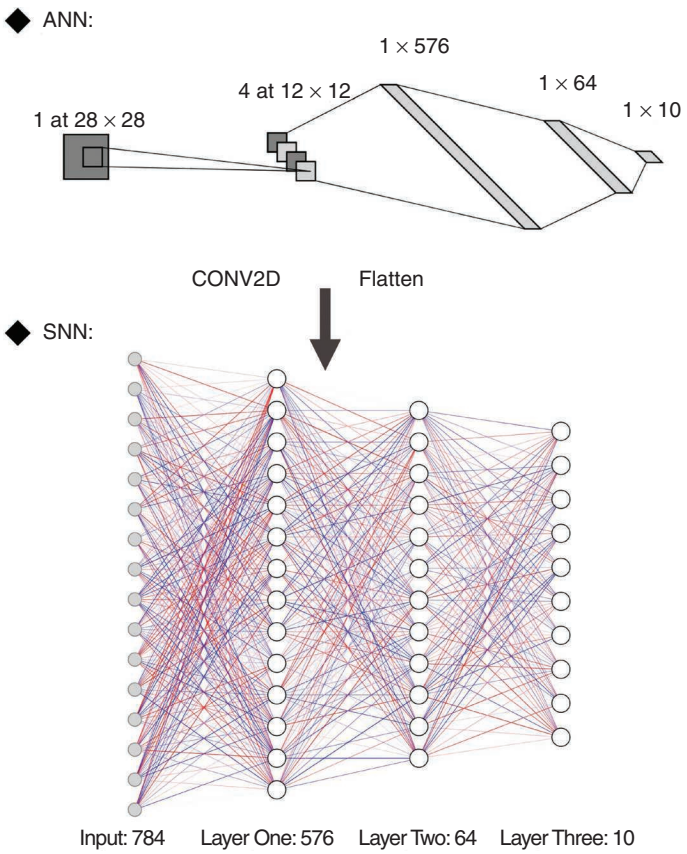

FIGURE 7. ANN and SNN model structures. CONV2D: convolution 2D.

structure. Another method is for an irregular connection. The connection number of neurons at the same layer may vary significantly. The first adaptation algorithm is used to calculate the number of nodes needed to use as few nodes as possible. The problem can be transformed into a packing problem. Each node, regarded as a box, puts neurons and corresponding connections. The box will be full when the number of neurons reaches the threshold, or another neuron's connections cannot be put. According to the data flow, we reverse the allocation of resources and gain an equal, one-to-one network corresponding to the network and hardware neurons. In the mapping process, a greedy algorithm can be employed to optimize power consumption.

CASE STUDY

Based on Darwin Mouse, we take the Modified National Institute of Standards

and Technology-based, handwritten numeral recognition model to illustrate the whole process of an SNN model's development, compilation, and execution on a brain-inspired computer, as illustrated in Figure 6.

In this case, the SNN model is developed by conversion. Initially, a classification ANN is constructed and trained using the traditional ANN framework TensorFlow. The weight parameters in the ANN are then optimized by means of parameter adjustment based on data-based weight normalization. Finally, using a model-conversion tool, the ANN is converted into the SNN. The algorithm it uses can be customized in line with user requirements.

We use the method presented by Diehl et al.[18] to convert a simple convolutional ANN into an SNN. The output SNN contains four layers, 784 virtual input nodes, 650 computational neurons, and 41,600 synapses, as depicted in Figure 7. The simulation time step is set to 1 ms, and the recognition time window is 100 ms. Finally, it takes a total of 19.123 s to convert the ANN into the SNN using our model-conversion tool.

Before the model is deployed to the actual brain-inspired computer, it is simulated by the simulator to verify the correctness and performance of the model. First, the number and type of neurons in each layer and the connection weight between layers are parsed from the model file described by DarwinMDL, and the SNN is created and initialized. Then, it is put on the simulator to run. After several simulation runs, the network parameters are optimized according to the results to improve the performance of the model.

After verifying the SNN on the simulator, the compiler compiles the DarwinMDL file, generates the executable model file, and submits it to the DarwinOS to be deployed on Darwin Mouse for operation. Then, under the control of DarwinOS, the image to be recognized is encoded into spikes and input into the computer, and the corresponding output is decoded to obtain the final result. The debugging tool provided by DarwinIDE is used to display the neural resource status and neural task execution results, as shown in Figure 8.

After working through the workflow described previously, our final result is as follows: On average, it took 56.063 ms to process an image. The SNN achieved 92.8 and 83% accuracy while running on the simulator and

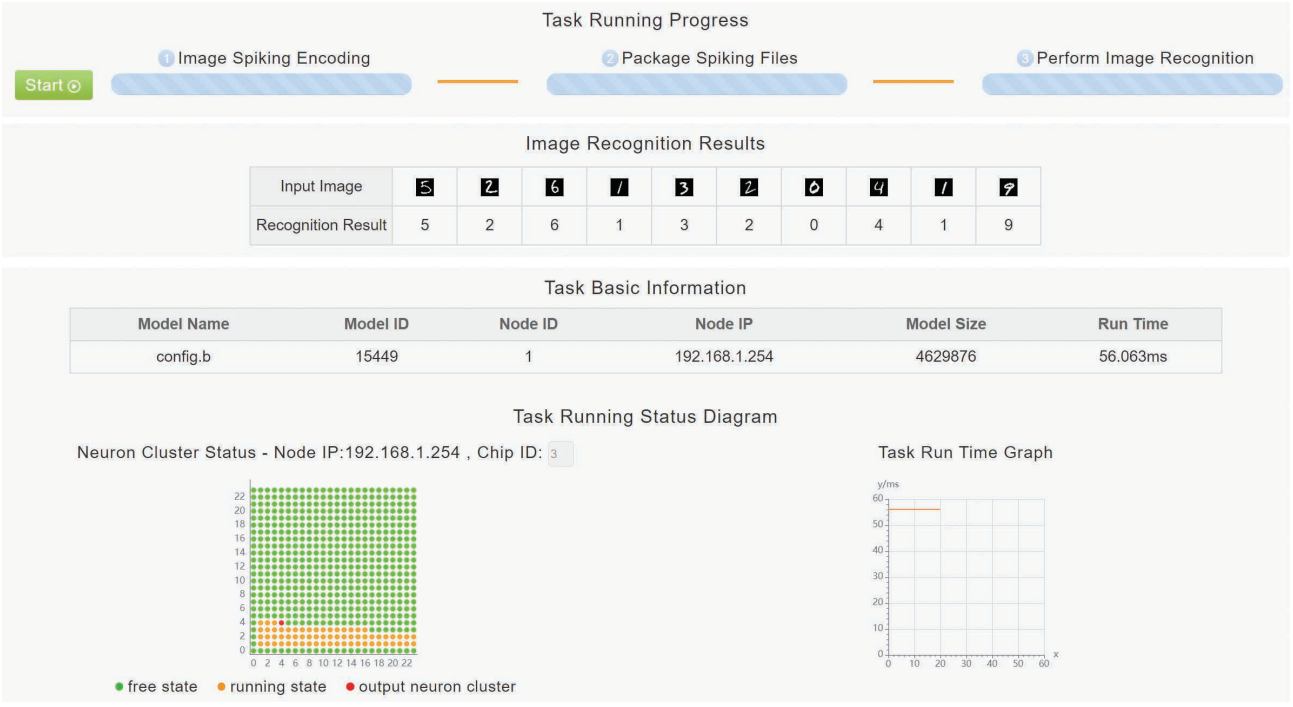

FIGURE 8. The state monitoring and execution result interface of DarwinIDE.

ABOUT THE AUTHORS

SHUIGUANG DENG is a full professor in the College of Computer Science and Technology, Zhejiang University, Hangzhou, 310027, China. His research interests include edge computing, service computing, and cloud computing. Deng received a Ph.D. in computer science from Zhejiang University. He serves as an associate editor for *IEEE Transactions on Services Computing, Knowledge and Information Systems, Computing,* and *IET Cyber-Physical Systems: Theory & Applications*. He is a Senior Member of IEEE and a fellow of the Institution of Engineering and Technology. Contact him at dengsg@zju.edu.cn.

PAN LV is a doctoral student at Zhejiang University, Hangzhou, 310027, China. His main research interests include intelligent systems, embedded real-time systems, and brain-inspired computing. LV received an M.Sc. in computer science and technology from Zhejiang University. Contact him at lvp@zju.edu.cn.

OUWEN JIN is a doctoral student at Zhejiang University, Hangzhou, 310027, China. His main research interests include computer architecture and brain-inspired computing. Jin received a bachelor's degree in computer science from the University of Electronic Science and Technology. Contact him at jinouwen@zju.edu.cn.

SCHAHRAM DUSTDAR is a professor of computer science with the Distributed Systems Group, TU Wien, Vienna, Austria. His research interests include service-oriented architectures and computing, mobile and ubiquitous computing, complex and adaptive systems, and context-aware computing. Dustdar received a Ph.D. in business informatics from the University of Linz, Austria. He is an elected member of Academia Europaea, where he is chairman of the Informatics Section. He serves as coeditor in chief of *ACM Transactions on Internet of Things* and editor in chief of *Computing*. He also serves as an associate editor of *IEEE Transactions on Services Computing, IEEE Transactions on Cloud Computing, ACM Transactions on the Web,* and *ACM Transactions on Internet Technology*. He serves on the editorial boards of *IEEE Internet Computing* and *Computer*. He is a Fellow of IEEE. Contact him at dustdar@dsg.tuwien.ac.at.

YING LI is an associate professor in the College of Computer Science, Zhejiang University, Hangzhou, 310027, China. His research interests include compiler techniques, service computing, and data science. Li received a Ph.D. in computer science from Zhejiang University. Contact him at cnliying@zju.edu.cn.

DE MA is an associate professor in the college of Computer Science and Technology, Zhejiang University, Hangzhou, 310027, China. His research interests include neuromorphic hardware, VLSI design, and system-on-chip architecture. Ma received a Ph.D. in electrical and information engineering from the Institute of VLSI Design, Zhejiang University. Contact him at made@zju.edu.cn.

ZHAOHUI WU is a professor of computer science at Zhejiang University, Hangzhou, 310027, China. His current research interests include intelligent systems, brain–machine interfaces, and service computing. Wu received a Ph.D. in computer science from Zhejiang University. He is a Fellow of IEEE, a fellow of the World Academy of Sciences, and a member of the Chinese Academy of Sciences. Contact him at wzh@zju.edu.cn.

GANG PAN is a professor in the College of Computer Science and Technology, Zhejiang University, Hangzhou, 310027, China. His research interests include neuromorphic computing, brain–machine interfaces, and artificial intelligence. Pan received a Ph.D. in computer science from Zhjiang University. He is the corresponding author of this article. Contact him at gpan@zju.edu.cn.

brain-inspired computer, respectively. Compared with the 93.27% accuracy obtained by the trained ANN fed into a model-conversion tool, the accuracy loss was 0.47 and 10.27%, respectively. The result is a little worse than the state-of-the-art method.[15] However, considering we did not specifically optimize the training and conversion process, and most of the parameters are just set defaults, we consider this result acceptable.

When it comes to brain-inspired computer software architectures, although academic and industrial circles have not reached a unified understanding of the standard, throughout existing brain-inspired computing systems, the direction of efforts remains the same. The abstract hardware details are expected to construct the primary software runtime environment and further establish the programming development environment and OS to simplify the programming method. In the future, brain-inspired computer software systems will gradually realize standardization from such aspects as the OS, programming language, and programming paradigm to promote the rapid and benign development of a brain-inspired computing ecology.

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Program of China (No.2021YFB2501300), the National Natural Science Foundation of China (No.U20A20220), and the National Important Science & Technology Specific Projects (No.2017ZX01038201).

REFERENCES

1. K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019, doi: 10.1038/s41586-019-1677-2.
2. S. Scholze *et al.*, "A 32 GBit/s communication SoC for a waferscale neuromorphic system," *Integration*, vol. 45, no. 1, pp. 61–75, 2012, doi: 10.1016/j.vlsi.2011.05.003.
3. 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, 2013, doi: 10.1109/JSSC.2013.2259038.
4. F. Akopyan *et al.*, "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1537–1557, 2015, doi: 10.1109/TCAD.2015.2474396.
5. M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018, doi: 10.1109/MM.2018.112130359.
6. J. Pei *et al.*, "Towards artificial general intelligence with hybrid Tianjic chip architecture," *Nature*, vol. 572, no. 7767, pp. 106–111, 2019, doi: 10.1038/s41586-019-1424-8.
7. D. Ma *et al.*, "Darwin: A neuromorphic hardware co-processor based on spiking neural networks," *J. Syst. Archit.*, vol. 77, pp. 43–51, Jun. 2017, doi: 10.1016/j.sysarc.2017.01.003.
8. E. Müller *et al.*, "The operating system of the neuromorphic brainScaleS-1 system," 2020, *arXiv:2003.13749*.
9. S. B. Furber *et al.*, "Overview of the SpiNNaker system architecture," *IEEE Trans. Comput.*, vol. 62, no. 12, pp. 2454–2467, 2013, doi: 10.1109/TC.2012.142.
10. A. G. D. Rowley *et al.*, "SpiNNTools: The execution engine for the SpiNNaker platform," *Frontiers Neurosci.*, vol. 13, pp. 231–231, Mar. 2019, doi: 10.3389/fnins.2019.00231.
11. J. Sawada *et al.*, "TrueNorth ecosystem for brain-inspired computing: Scalable systems, software, and applications," in *Proc. Int. Conf. High Perf. Comput., Netw., Storage Analy.*, 2016, pp. 130–141, doi: 10.1109/SC.2016.11.
12. J. A. Pérez-Carrasco *et al.*, "Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing--Application to feedforward ConvNets," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 11, pp. 2706–2719, 2013, doi: 10.1109/TPAMI.2013.71.
13. Z. Bing, I. Baumann, Z. Jiang, K. Huang, C. Cai, and A. Knoll, "Supervised learning in SNN via reward-modulated spike-timing-dependent plasticity for a target reaching vehicle," *Frontiers Neurorobot.*, vol. 13, p. 18, May 2019, doi: 10.3389/fnbot.2019.00018.
14. B. Rueckauer and S.-C. Liu, "Conversion of analog to spiking neural networks using sparse temporal coding," in *Proc. 2018 IEEE Int. Symp. Circuits Syst.*, pp. 1–5, doi: 10.1109/ISCAS.2018.8351295.
15. 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, doi: 10.3389/fnins.2017.00682.
16. Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *Int. J. Comput. Vis.*, vol. 113, no. 1, pp. 54–66, 2014, doi: 10.1007/s11263-014-0788-3.
17. P. Gu, R. Xiao, G. Pan, and H. Tang, "STCA: Spatio-temporal credit assignment with delayed feedback in deep spiking neural networks," in *Proc. 2019 Int. Joint Conf. Artif. Intell.*, pp. 1366–1372, doi: 10.24963/ijcai.2019/189.
18. P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *Proc. 2015 Int. Joint Conf. Neural Netw.*, pp. 1–8, doi: 10.1109/IJCNN.2015.7280696.