Towards a Unified Architecture for Reproducible Neuromorphic Research

Introduction

Reproducibility in computational neuroscience and neuromorphic computing is increasingly critical. Traditional academic papers often fall short in enabling others to **verify and re-run simulations** due to incomplete method descriptions and complex dependencies [1]. The community faces a *reproducibility crisis*, where sharing code and data is not enough if the research cannot be executed easily by readers [1]. To address this, there is a push toward **executable papers** – publications that come with ready-to-run code, data, and computational environments. This report explores how current and future neuromorphic computing frameworks and simulators (such as NEST, Brian2, SpiNNaker, Loihi, etc.) can integrate into a **unified architecture** for reproducible, executable research. Key elements of this unified approach include standardized publication formats (e.g. Jupyter notebooks, RO-Crate research object packages), containerized environments, accessible data, and encapsulated execution workflows. We review existing practices and initiatives, identify technical gaps, and provide actionable recommendations to benefit the global neuroscience and neuromorphic computing research communities.

Current Practices in Reproducible Computational Neuroscience

Interactive Notebooks and Executable Articles: An emerging practice is to use Jupyter Notebooks or similar literate programming tools to capture code, results, and narrative together. For example, *eLife* now supports Executable Research Articles (ERA) which let authors embed live code, data, and interactive figures directly in papers [2] [3]. Readers can modify and re-run code in the browser to explore how analyses were done [4]. This bridges the gap between a static paper and the dynamic computational workflow by treating code and data as *first-class citizens* in publications [5]. Open-source tools like Stencila underpin such efforts, allowing Jupyter notebooks or R Markdown to be integrated with journal articles [3]. These practices make results more transparent and allow researchers to *inspect, adjust, and re-execute* analyses easily [4].

Containerization and Environment Capture: Reproducibility is bolstered by packaging the software environment using containers (e.g. Docker or Singularity) or virtual machines. By distributing a pre-configured container image, researchers ensure that the exact operating system, libraries, and simulator versions are preserved [6]. This relieves readers from "dependency hell" – the pain of assembling the correct versions of each tool [7]. For instance, the NEST simulator team provides an official Docker image of NEST [8], and community guides show how to create Dockerfiles bundling tools like NEST, NEURON, Brian, PyNN, and common Python libraries [9]. Such images allow others to launch the same simulation environment with a single command, either locally or via cloud services. Initiatives like **Binder** enable hosting code repositories with an `environment.yml` or `Dockerfile` so that anyone can launch the code in a web-based Jupyter instance, further lowering barriers to execution. In practice, many researchers now include a `requirements.txt`, Conda environment file, or Dockerfile with their code to facilitate environment setup, moving beyond mere code sharing to *executable environment sharing* [7].

Standardized Data and Model Formats: Standard formats play a role in making results reproducible and interoperable. In computational neuroscience, the **NeuroML** model description language is a community standard for specifying neuron and network models in a simulator-independent way [10]. Tools that support NeuroML allow models to be exchanged between simulators, enhancing reproducibility and reusability of model definitions. Similarly, the **PyNN** API provides a common Python interface to define spiking neural networks that can run on multiple back-end simulators (NEST, NEURON, Brian, etc.) [11]. By writing a model once in PyNN, researchers can execute it on different simulators or even neuromorphic hardware without code changes [11]. This encourages cross-validation of results and ensures that model code is not tied to one proprietary framework, addressing fragmentation. Other data standards, such as **Neurodata Without Borders (NWB)** for neural data, and use of persistent identifiers (DOIs) for datasets, further contribute to making data **accessible and verifiable**. Current best practices include depositing datasets in open repositories (Zenodo, OSF) and citing them via DOIs, so that published code can automatically download required data if not packaged [12]. Taken together, these practices — interactive notebooks, containerization, and standard formats — form the groundwork of reproducible workflows in neuroscience today.

Neuromorphic Frameworks and Simulator Capabilities

A variety of spiking neural network simulators and neuromorphic computing platforms are used in the community. Each has distinct capabilities, and understanding how they can plug into a reproducible workflow is key. **Table 1** summarizes several prominent frameworks and their features related to reproducibility and interoperability:

Framework	Description & Key Capabilities	Standards Compatibility	Reproducibility & Integration
NEST (NEural Simulation Tool)	Open-source SNN simulator (C++ core, with PyNEST Python API). Focuses on large-scale networks and precise spike timing dynamics. Supports 50+ neuron models and many synapse types; proven to handle millions of neurons with billions of connections in HPC environments [13]. Excellent multi-threading and MPI parallelization for high-performance simulations [14].	Supports **PyNN** common interface [11] (write model once, run on NEST or others). Model import/export via **NeuroML** is possible through conversion tools and community standards alignment [10]. Interoperable with other tools (e.g., via MUSIC for co-simulation).	Python API allows use in Jupyter notebooks. Official Docker images available for consistent environments [8]. Continuous integration tests ensure simulation correctness and **reproducible results** across versions [15]. NEST's minimal dependencies and open-source license further ease integration [16] [17].

Framework	Description & Key Capabilities	Standards Compatibility	Reproducibility & Integration
Brian2	Open-source SNN simulator in pure Python, emphasizing **ease of use and flexibility** for model building [18]. Uses a user-friendly syntax where neuron models are defined by mathematical equations, lowering the barrier for new modelers [19]. Can generate optimized C++ or CUDA code for efficiency while maintaining interactivity. Well-suited for smaller to mid-scale models and rapid prototyping in research and teaching.	Supports **PyNN** API (Brian is one of the back-end simulators for PyNN) [11], allowing Brian models to be specified in a standard way. Tools exist to export/import models via **NeuroML**, though not native. Strong alignment with Python scientific stack (NumPy, etc.) enables integration with analysis workflows.	Easily installed via pip/conda on any platform [20]; thus straightforward to include in containers or Binder environments. Brian emphasizes backwards compatibility (older scripts continue to run) and consistent results across versions [21]. It provides controlled random number seeding for reproducible simulations [22]. Its pure-Python nature means researchers can embed Brian simulations in notebooks and **share the exact code** used to generate figures with minimal friction.

Framework	Description & Key Capabilities	Standards Compatibility	Reproducibility & Integration
SpiNNaker (and **sPyNNaker** software)	Specialized neuromorphic hardware platform (many-core ARM processors) for **real-time, large-scale spiking network simulation** [23]. A SpiNNaker machine can simulate millions of neurons in biological real time by distributing computation across thousands of cores, with a focus on parallel event-driven communication. SpiNNaker hardware is used for brain model emulation (e.g. a full-scale cortical microcircuit).	Programmed via the standard **PyNN** API (through the sPyNNaker library) [11], meaning the *same PyNN script* that runs on NEST or Brian can run on SpiNNaker hardware [11]. This abstraction fosters reproducibility: researchers can validate a model in software then deploy to hardware with minimal changes. Data formats and tools from software simulators (e.g., for analyzing spike trains) are reused.	The PyNN interface and Python client code can be containerized, but running on actual hardware requires access to a SpiNNaker board or cluster (available via the Human Brain Project's EBRAINS platform [24]). In a reproducible workflow, one strategy is to **fall back to software simulators** for those without hardware – e.g. run the PyNN model in NEST for basic result replication, and note hardware-specific differences. Alternatively, cloud-hosted SpiNNaker resources (such as university or EBRAINS servers) could be linked. Ensuring reproducibility here means providing *both* the code and either an accessible hardware endpoint or a simulated mode. Containerizing the software stack (sPyNNaker, its dependencies, ARM toolchains) is possible, but less common. Continued development (SpiNNaker 2) promises higher capacity and may come with improved remote access, making hardware-in-the-loop publishing more feasible.

Framework	Description & Key Capabilities	Standards Compatibility	Reproducibility & Integration
Intel Loihi (with **Lava** framework)	Loihi 1 & 2 are neuromorphic research chips from Intel, featuring many cores of spiking neurons with on-chip learning engines. Loihi 2 (2021) supports up to ~1 million neurons per chip and introduces more flexible spike messaging and embedded x86 cores for management [25] [26]. The new *Lava* software framework is an open-source, platform-agnostic library for developing neuromorphic applications, not limited to Loihi hardware [27]. Lava provides a high-level **neuro-inspired programming model** that can target Loihi or run in simulation on CPUs, making it a bridge between software and neuromorphic silicon.	Lava is open-source and in principle could be extended to support standards, but currently **does not use PyNN** or NeuroML out-of-the-box (it defines its own abstractions). However, being Python-based and modular, it could interoperate with other tools (for example, converters could translate PyNN models to Lava Processes, or export Lava networks to NeuroML in the future). The focus is on platform-independent design, which complements standardization efforts.	With *Lava*, Intel signaled a commitment to open ecosystems, which aids reproducibility. Researchers can share Lava code, and because Lava can run a CPU simulation, reviewers don't need the Loihi chip to execute the code. This CPU fallback is crucial for reproducible publishing. The environment (Lava, Python libraries) can be encapsulated in a Docker container or Conda environment, enabling others to run the experiments without specialized hardware. Intel's Neuromorphic Research Cloud also allows authorized users to run code on Loihi 2 hardware remotely [28] – a potential model for executing published neuromorphic experiments on-demand. As Lava matures, we expect containerized *reference environments* to be published (community Docker images for Lava exist or can be created) so that each Loihi experiment can be packaged with its exact software stack.

Table 1: *Capabilities of key neuromorphic simulation frameworks and their compatibility with reproducible workflow standards.* Each framework's design impacts how easily it can be integrated into a standardized, **executable publication** pipeline.

As shown above, many tools already embrace features that support reproducible research: Python interfaces for scriptability, support for common standards (PyNN, NeuroML), and open-source availability. Notably, PyNN serves as a unifying layer across NEST, NEURON, Brian, and SpiNNaker [11], encouraging a culture of writing simulator-agnostic model code. In practice, this means a researcher can run identical model code on multiple backends, helping verify that results are not an artifact of one simulator's implementation [29]. For example, a study reported running the same network on NEST and NEURON via PyNN and observing very similar spike outputs, with divergences only arising from expected chaotic dynamics [30] [29]. Such cross-validation would be impossible without standard interfaces.

From a *publication* standpoint, frameworks that are easily pip-installable or have Docker support (like NEST, Brian, and now Lava) are straightforward to embed in a reproducible workflow. Authors can include a requirements file listing `nest-simulator`, `brian2`, etc., or base their container on an image that has these installed [9]. Indeed, community containers have been built bundling multiple simulators and tools for convenience [9]. An example noted by the NeuralEnsemble project was a Docker image pre-loaded with NEST, NEURON, Brian, PyNN, and common libraries, so that a wide array of computational neuroscience models could be run out-of-the-box [9]. This suggests that a **unified reproducible architecture** might include a *standard container or environment* that each tool can plug into.

Technical Gaps and Challenges

Despite progress, there remain significant gaps in achieving a seamless, unified reproducible publishing architecture:

- **Fragmentation of Formats and Workflows:** Researchers use a patchwork of approaches – some share Jupyter notebooks, others provide scripts with manual setup instructions, others only share code on GitHub. There is no universally adopted *standard format* for an "executable paper". While Jupyter-based ERAs and tools like Code Ocean or RunMyCode exist, these are not yet the norm. A unified approach (such as the proposed JEN Reproducibility Bundle) is still in early adoption. This means readers face inconsistent experiences and often need to invest effort to run different papers' code.

- **Dependency and Environment Complexity:** Containerization is powerful, but not all authors are familiar with creating Dockerfiles or managing complex dependencies. Sometimes environments involve proprietary drivers (GPU, FPGA or neuromorphic hardware interfaces) that are harder to containerize or require specific host support. For neuromorphic hardware, one challenge is that **hardware-in-the-loop reproducibility** is not straightforward – e.g., running a Loihi experiment exactly may require access to Loihi boards or cloud service, which is not as simple as running a software simulation. Emulators or fallbacks help, but differences in hardware vs software execution (e.g., real-time behavior) might affect results.

- **Lack of Incentives and Culture:** Until very recently, there has been little academic reward for investing time in reproducible infrastructure. Writing Dockerfiles, curating metadata, and preparing interactive notebooks is extra work not explicitly rewarded in traditional publishing. The academic incentive structure favored novel results over reproducible ones [31] [32]. This is a cultural gap that initiatives like JEN's Executable Citation Index aim to address by giving citation credit to the published computational artifacts themselves [33] [34]. However, as it stands, not all projects put equal emphasis on reproducibility, leading to variable quality in supplementary materials.

- **Standards Integration in All Tools:** Not every framework fully supports community standards. For instance, not all simulators can import NeuroML models, and PyNN, while powerful, might not cover some newer frameworks (e.g., PyNN does not yet have a backend for Loihi/Lava). This means a "unified" architecture has to either extend these standards or develop adapters. The lack of a common *experiment configuration schema* is another gap – researchers often hard-code parameters in scripts, whereas a more standardized approach would separate configuration (e.g. a YAML/JSON of parameters) from code [35].

- **Data Management and Longevity:** Ensuring long-term access to data and code is non-trivial. Links can break, repositories can disappear. While DOIs for data and services like Zenodo or OSF mitigate this, not all authors use them. Also, large neuroscience datasets (imaging data, large spike datasets) cannot be easily bundled with a paper. The reproducible architecture must handle external data sources robustly (for example, by providing scripts to download data with checksums [12]). There is also the issue of **software rot** – code that relies on specific old library versions might not run in the future. Without encapsulating the environment (via containers and metadata like RO-Crate), reproducibility degrades over time.

- **Execution Infrastructure:** Even if an executable paper provides all pieces, the reader needs somewhere to run it. Not all readers have the computational resources to reproduce large models (e.g., requiring HPC or neuromorphic hardware). Cloud-based sandboxing (like Binder or journal-provided execution servers) is a solution but not widely available for all publications yet. There is a gap in infrastructure to *automatically* execute and verify submitted code in the publication pipeline. Some journals lack the technical backend to support execution during review or by readers. This technical gap is being addressed by pilot platforms (eLife ERA uses a cloud container execution service; JEN proposes a CI/CD pipeline on submission [36] [37]), but it's not universally present.

In summary, the field needs more consistency in how reproducible artifacts are packaged, better integration of hardware-dependent experiments, and stronger incentives and infrastructure to support the executable research paradigm.

Existing Initiatives and Standards Paving the Way

Encouragingly, several initiatives are actively working toward solutions:

- **Journal of Executable Neuroscience (JEN) – Reproducibility Bundles:** The JEN (currently in planning) proposes that every accepted paper **must include a Reproducibility Bundle** – a complete archive with code, data, and environment to regenerate results [38] [39]. The JEN bundle specification defines five required components: (1) a declarative experiment configuration file for parameters [40], (2) all necessary data (or scripts/DOIs to fetch it) [12], (3) an environment container definition (Dockerfile) capturing the exact software stack [7], (4) raw run artifacts (e.g. raw output data like spike times, logs) [41], and (5) visualization/analysis scripts to regenerate every figure from those raw outputs [42]. By enforcing this structure, JEN aims to shift the default from static papers to *verifiable computational artifacts*. Notably, JEN plans to adopt **RO-Crate** as the packaging standard for these bundles [43] [44]. Each bundle will include a machine-readable `ro-crate-metadata.json` describing its contents (code, data, config, etc.), ensuring longevity and interoperability [45]. This means even if technology changes, the metadata will explain how to run the experiment. JEN also aligns with existing best practices, e.g. embracing NeuroML for model descriptions and tools like Sumatra for provenance tracking [46] instead of reinventing the wheel. Although JEN is a neuroscience-focused effort, its standards could serve as a blueprint for other fields.

- **Executable Research Articles (ERA) by eLife:** As mentioned, eLife's ERA is a pioneering implementation in a general journal. It allows published papers to be enriched with live code blocks. The underlying technology (initially developed as the *Reproducible Document Stack* with Stencila) is open-source and designed to be adopted by other publishers [47] [3]. In an ERA, readers can run analysis code on the fly and even tweak it within the article, with all changes sandboxed to their

session [4]. Importantly, the ERA package (which includes the notebook, data, and environment specs) can be downloaded by readers for offline reuse [48]. This directly addresses the "last mile" of reproducibility – making it *trivial for the reader* to engage with the actual computations of the paper. The success of ERA in eLife has provided a real-world example that executable papers are feasible and valued by researchers [49]. It also demonstrated that authors are willing to do the extra work when given the tools and platform support.

- **NeuroML and Open Source Brain:** The **NeuroML ecosystem** has grown as a standard for multi-scale neural modeling, enabling model sharing between tools [50]. Open Source Brain (OSB) is a platform that hosts neuroscience model repositories and provides interactive cloud workspaces to simulate them. OSB explicitly aims to improve *accessibility, transparency, and reproducibility of models* and encourages reuse by the community [51]. Through OSB, one can launch JupyterLab or other apps with pre-installed neuroscience libraries to run simulations on shared models [52] [53]. For example, OSB integrates with Jupyter and NetPyNE, and allows running NeuroML models in the browser via back-end simulators [52] [53]. This kind of platform shows how a repository of models can be coupled with execution environments, hinting at the architecture needed for reproducible papers: a combination of **cloud-accessible code repositories + on-demand computing**. While OSB is model-focused, not a journal, it complements publishing initiatives by ensuring models from papers can be readily examined and run by others.

- **The Human Brain Project's EBRAINS platform:** EBRAINS (a European infrastructure from the Human Brain Project) provides tools for collaborative research, including a Neuromorphic Computing service. Researchers can upload their code and run it on neuromorphic hardware like SpiNNaker or BrainScaleS through a web interface or via Jupyter notebooks. This kind of integrated platform can be leveraged for reproducible research: a published paper's code could, for example, be packaged as an EBRAINS "collaboratory" so that readers with an account can execute the exact experiment on the same hardware used by the authors. EBRAINS has also adopted standards like PyNN for its workflows (e.g., **sPyNNaker is available as an EBRAINS tool** [24]). Although not a publication venue, EBRAINS illustrates how providing broad access to specialized hardware can be done in a reproducible way (with version-controlled scripts, Jupyter environments, etc., all accessible).

- **Community-driven Reproducibility Guides:** Projects like **The Turing Way** (an open guide to reproducible data science) are influencing practices in neuroscience by providing how-to instructions for reproducibility (version control, environment management, testing, etc.) [46]. Likewise, tools such as **Sumatra** (for provenance tracking of simulations) have been used in computational neuroscience to record the exact code version, parameters, and environment for each run, so results can be traced and repeated [46]. These tools are building blocks that can be integrated into a unified workflow. For example, Sumatra or DVC can track data and code changes; continuous integration services can automatically run test simulations whenever code is updated (some simulator developers already do this – NEST's CI ensures each commit passes a testsuite for correct results [54]).

- **ReScience and Replication Journals:** *ReScience C* is an example of a journal that publishes replicability studies, where authors re-implement and validate old results. While not focusing on new neuromorphic methods, it emphasizes executable submissions (the code is peer-reviewed alongside the article). This trend reinforces the idea that code and paper are inseparable for scientific

communication in computational fields. JEN's plan to include *Replication Reports* as a category builds on this, giving formal recognition to reproduction efforts [55] [56].

Collectively, these initiatives indicate momentum toward *reproducible publishing*. They provide pieces of the architecture: standard formats (NeuroML, RO-Crate), interactive execution platforms (ERA, OSB, EBRAINS), packaging tools (Docker, Binder), and new publishing standards (JEN, ReScience). The challenge and opportunity now is to synthesize these into a coherent, unified architecture that can be widely adopted.

Recommendations for a Unified Reproducible Architecture

Bringing the above threads together, we propose an architecture and best practices that would enable neuromorphic frameworks to **plug into a standardized reproducible publishing workflow**. The following recommendations serve as a blueprint:

- **1. Define a Standard "Executable Paper Bundle" Format:** Establish a discipline-wide standard (building on RO-Crate and the JEN Reproducibility Bundle concept) for what authors submit alongside a manuscript. This should include:
- **Code and Scripts:** All simulation code, analysis scripts, and notebooks needed to produce the results. Where possible, provide *literate* scripts (with narrative or notebook format) to enhance clarity.
- **Environment Specification:** A Dockerfile or container image reference, plus a plain text environment file (e.g. `environment.yml` for Conda) for transparency [7]. This ensures anyone can reconstruct the software setup. The container should ideally be based on a common baseline (e.g., a community-provided image with popular neuroscience libraries, to reduce build complexity [9]).
- **Configuration & Metadata:** A human-readable config file (YAML/JSON) for experimental parameters [35] and a machine-readable metadata file (RO-Crate `ro-crate-metadata.json`) describing the contents of the bundle [44]. The metadata should list the files (code, data), their roles, and how to invoke the experiment (e.g., "run `python simulate.py --config params.yaml` to reproduce Figure 1") [57].
- **Data and Outputs:** Either include all small datasets or provide scripts + DOIs for downloading large data [12]. Also, include representative raw output files (e.g. spike times, log files) that were generated by the code [41]. This allows validation that running the code yields identical outputs. For large outputs, guidelines could suggest how to store them (perhaps on Zenodo, with the DOI in the metadata).

- **Result Regeneration Scripts:** Include scripts or notebook sections that take the raw outputs and regenerate every figure and table in the paper [42]. This "closing the loop" step is crucial – it demonstrates that the analysis from data to figures is automated and captured. Reviewers or readers can run this to verify the published figures match the code's output [58].

- **2. Modularize and Encapsulate Each Simulator's Role:** Neuromorphic tools should be packaged in a way that they can be **swapped or updated easily** within the workflow. For example, if using PyNN, the code can target NEST for one run and SpiNNaker for another – authors should exploit this to show reproducibility across platforms when relevant. We encourage framework developers to maintain Docker images or Conda packages for their tools (many already do) and to provide sample integration code for reproducible workflows. For instance, NEST could provide a template repository demonstrating how to write a paper's simulation in PyNEST and bundle it with a Dockerfile and

metadata. Brian2, being pure Python, can integrate by ensuring all features are accessible programmatically (which they are) and by continuing to emphasize backward compatibility so old scripts don't break [21]. For hardware-dependent platforms like SpiNNaker or Loihi, the recommendation is two-fold:

- Provide a **software-emulation mode** or compatibility layer. Lava already allows CPU simulation of Loihi models – this should be a standard practice for any neuromorphic hardware. SpiNNaker's use of PyNN means the exact model could run in NEST if needed (albeit not in real time). Authors should include instructions or config options for such a fallback, enabling readers without hardware to still execute the model (perhaps at smaller scale or speed).

- Develop **remote execution hooks**. Journals or platforms could partner with hardware providers (e.g., Intel INRC for Loihi, University of Manchester for SpiNNaker) to allow readers to run the code on actual hardware via the cloud. This could be as simple as providing an API endpoint or a lightweight web interface where the Reproducibility Bundle can be deployed. While not trivial, this is analogous to how some astronomy journals provide links to remote telescopes or databases. A successful example is how Intel's Neuromorphic Cloud let researchers time-share Loihi hardware [28] – a paper could leverage that by including a script that offloads part of the experiment to that service if available.

- **3. Embrace Containerized Execution and CI/CD in Publishing:** We recommend that the publication workflow include automated **continuous integration checks** on the submitted bundles. This means when an author submits the executable bundle, the journal (or an associated platform) automatically attempts to build the container and run the provided test commands. If any step fails (container won't build, tests don't pass, figures don't match), the submission is flagged for revision [37] [59]. This CI pipeline acts as a "gatekeeper" to enforce reproducibility standards before a human even reviews the science [60] [61]. JEN's plan details such a pipeline: build the Docker, run the experiment, check outputs vs. manuscript figures, etc., returning a report to authors [59] [58]. We strongly support this approach, as it shifts the burden of testing from the reader to the publishing process itself. It also encourages authors to use best practices (like deterministic random seeds, thorough documentation) to pass the automated checks. Reproducibility becomes a first-class criterion for acceptance, not an afterthought.

- **4. Leverage and Extend Standards for Interoperability:** To reduce technical friction, all simulators and tools should **adhere to or support common standards**. PyNN should be extended to new frameworks (the community could develop a PyNN interface for Loihi/Lava, for example). Likewise, simulators could offer import/export of NeuroML or other model description standards so that models from one tool can be validated on another. This addresses model reusability: an experiment's model could be re-run on a different simulator in the future to compare results or when hardware becomes obsolete. Another key standard is data format – using HDF5 or NWB for output data can make it easier to share large result files and analyze them with common tools. The **NeuroML/LEMS** ecosystem, which allows defining complete experiments in an XML-based format, could serve as a lingua franca: a published model in NeuroML can be executed on any compatible simulator (potentially via OSB's browser-based simulators [62]). Thus, we recommend authors (whenever feasible) to provide a NeuroML (or other standard) version of their model in addition to the native code. Even if the primary results used custom code, a standard representation aids longevity. Journals could incentivize this by, e.g., having a badge or note for "Model also available in NeuroML".

- **5. Ensure Metadata and Accessibility for the Long Term:** Each executable paper (and its bundle) should be registered with persistent identifiers. For example, deposit the entire bundle in Zenodo (which grants a DOI) upon publication [63]. This way, even if journal websites reorganize, the artifact remains accessible. The metadata (using RO-Crate) will describe how to use the bundle even decades later [45]. We also advise including rich documentation in the bundle (README files, usage notes) to help future researchers. **Software Heritage** can be used to archive the source code in an independent archive as well [64]. By layering these measures – DOI, RO-Crate, and code archiving – we guard against bit-rot and link rot, ensuring reproducibility isn't lost to time.

- **6. Foster Community and Training:** Technology alone won't solve it; researchers need training in these practices. We should integrate lessons on reproducible workflows into neuromorphic computing curricula and workshops. Initiatives like Open Neuromorphic's community events (as noted in their collaborations between projects like Brian, GeNN, Nengo, etc. [65] [66]) can include hackathons to help research groups containerize their simulators or create example executable papers. Sharing success stories where a fully reproducible paper led to quicker adoption of a new tool (because users could immediately run the provided bundle) will motivate others. Indeed, a method paper that comes with an immediate runnable example (as JEN suggests for Methods/Framework papers [67]) likely accelerates community uptake of that method. Funding agencies and conference organizers could also reward reproducibility (e.g., "best reproducible paper" awards or requirements for supplemental code in neuromorphic research proposals). Over time, as these practices become routine, the whole field benefits from **faster verification, easier reuse of models, and more trust in computational results**.

Conclusion

Neuromorphic computing and computational neuroscience stand to gain enormously from a shift to fully reproducible, executable publications. By standardizing how we package and share our simulations – from the spiking network code down to the software environment and data – we enable any researcher to *press a button and rerun an experiment*, validating and building upon prior work. The frameworks we use today are more ready than ever for this shift: most are open source, have Python interfaces, and support common standards, making integration feasible. The gaps that remain are being actively addressed by forward-looking initiatives like JEN's reproducibility bundles and eLife's ERA, which show that executable science is not just idealistic, but practical and beneficial [5] [39].

In a unified architecture, each tool – be it NEST driving a large-scale cortical simulation, or Loihi hardware demonstrating a synaptic learning rule – becomes a *modular component* in a reproducible pipeline. Authors would spend a bit more effort upfront to containerize their work and document it, and in return, the community gets a permanent, verifiable artifact. Readers could mix and match components (for example, swap in a different simulator to test robustness), fostering collaboration. Reproducible publishing also accelerates learning: new students can literally run the paper to understand it, tweaking parameters to see effects, which turns every paper into a teaching resource.

The vision of an **executable neuroscience paper** standard is within reach. To realize it, we must continue to build bridges between tool developers, researchers, and publishers. Framework developers should prioritize interoperability and packaging, researchers should embrace the available tools and demand such standards from journals, and publishers should invest in the infrastructure to support execution and verification. By doing so, the field of neuromorphic computing can lead by example in the broader scientific

enterprise, showing how we can move from an era of merely *archival* publications to an era of *operational, reproducible science* [68]. The outcome will be a research culture where results are not just reported but actively tested and reused, driving faster and more reliable progress in our quest to understand and emulate neural systems.

Sources: The information and quotations in this report are drawn from a range of connected sources, including proposals for executable neuroscience publishing [67] [43], documentation of simulators and standards (NEST, Brian, PyNN, NeuroML) [11] [13], and commentary on emerging practices in open science and reproducible workflows [3] [46], as detailed in the reference brackets throughout the text. These illustrate both the state of the art and the roadmap toward the unified architecture described above.

---

1 7 10 12 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 55 56 57 58 59 60 61 63 64 67 68 Executable Neuroscience Journal Plan.pdf
file://file-ARHnsvVPcf4ki6APdGYk4K

2 3 4 5 47 48 49 eLife launches Executable Research Articles for publishing computationally reproducible results | For the press | eLife
https://elifesciences.org/for-the-press/eb096af1/elife-launches-executable-research-articles-for-publishing-computationally-reproducible-results

6 9 reproducibility_lightningtalks_codejam.key
https://neuralensemble.org/media/slides/reproducibility_lightningtalks_codejam.pdf

8 Docker — NEST Simulator Documentation - Read the Docs
https://nest-simulator.readthedocs.io/en/v3.8/installation/docker.html

11 Frontiers | sPyNNaker: A Software Package for Running PyNN Simulations on SpiNNaker
https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2018.00816/full

13 14 NEST - Open Neuromorphic
https://open-neuromorphic.org/neuromorphic-computing/software/snn-frameworks/nest/

15 16 17 54 Features — NEST simulator user documentation 3.0 documentation
https://nest-simulator.readthedocs.io/en/v3.1/features.html

18 19 65 66 Brian - Open Neuromorphic
https://open-neuromorphic.org/neuromorphic-computing/software/snn-frameworks/brian/

20 The Brian Simulator | The Brian spiking neural network simulator
https://briansimulator.org/

21 22 Compatibility and reproducibility — Brian 2 2.9.0 documentation
https://brian2.readthedocs.io/en/2.9.0/introduction/compatibility.html

23 Neuromorphic Hardware Guide - Open Neuromorphic
https://open-neuromorphic.org/neuromorphic-computing/hardware/

24 sPyNNaker - Tools - EBRAINS
https://www.ebrains.eu/tools/spynnaker

25 26 27 28 Intel Introduces 2nd Gen Neuromorphic Research Chip: Loihi 2 on Intel 4 EUV Process – WikiChip Fuse
https://fuse.wikichip.org/news/6383/intel-introduces-2nd-gen-neuromorphic-research-chip-loihi-2-on-intel-4-euv-process/

29 30 PyNN: A Common Interface for Neuronal Network Simulators - PMC
https://pmc.ncbi.nlm.nih.gov/articles/PMC2634533/

50 The NeuroML ecosystem for standardized multi-scale modeling in ...
https://elifesciences.org/articles/95135

51 52 53 Open Source Brain
https://www.opensourcebrain.org/

62 Simulating NeuroML Models
https://docs.neuroml.org/Userdocs/SimulatingNeuroMLModels.html