The Journal of Open Source Software

CANNs: Continuous Attractor Neural Networks Toolkit

Sichao He[1,2,3,4], **Aiersi Tuerhong**[5], **Shangjun She**[1,2,3,4], **Tianhao Chu**[1,2,3,4], **Yuling Wu**[1,2,3,4], **Junfeng Zuo**[1,2,3,4], **and Si Wu**[1,2,3,4¶]

1 School of Psychological and Cognitive Sciences, Peking University, Beijing, China **2** Peking-Tsinghua Center for Life Sciences, Academy for Advanced Interdisciplinary Studies, Peking University, Beijing, China **3** PKU-IDG/McGovern Institute for Brain Research, Peking University, Beijing, China **4** Center of Quantitative Biology, Peking University, Beijing, China **5** College of Mathematics and Statistics, Chongqing University, Chongqing, China ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 19 January 2026
Published: unpublished

Summary

CANNs (Continuous Attractor Neural Networks toolkit) is a Python library built on BrainPy, a powerful framework for brain dynamics programming. It streamlines experimentation with continuous attractor neural networks and related brain-inspired models. The library delivers ready-to-use models, task generators, analysis tools, and pipelines—enabling neuroscience and AI researchers to move quickly from ideas to reproducible simulations.

Statement of need

Continuous Attractor Neural Networks (CANNs) are a canonical neural circuit model for information processing in the brain and are receiving increasing attention in the fields of neuroscience and brain-inspired computing. They not only provide a mathematical model for understanding how the brain encodes continuous variables, such as spatial position, head direction, and moving direction, but also serve as a theoretical framework for elucidating how the brain processes abstract features and relationships. Known examples include that CANNs successfully explain key phenomena in hippocampal place cells (O'Keefe & Dostrovsky, 1971), entorhinal grid cells (Hafting et al., 2005), and head direction systems (Taube et al., 1990). Despite the importance, the CANN research suffers from fragmentation: researchers implement models from scratch, use incompatible codebases, and face significant reproducibility barriers. This lack of standardization slows progress and creates steep learning curves for newcomers.

CANNs toolkit addresses this gap by providing a unified Python toolkit built on a user-friendly and efficient programming framework BrainPy (Wang et al., 2023, 2025). It delivers: (1) standardized implementations of CANNs and related brain-inspired models, including mathematically tractable CANN models (Amari, 1977; Wu et al., 2008), adaptation-augmented CANN models (Li et al., 2025; Mi et al., 2014), grid cell networks (Burak & Fiete, 2009), alongside additional attractor architectures; (2) integrated task generation, simulation, and analysis pipelines; (3) high-performance computation via JAX JIT compilation and optional Rust acceleration. By standardizing the workflow—analogous to Hugging Face Transformers in deep learning—this library accelerates reproducible research and lowers barriers for computational neuroscientists, AI engineers, and students exploring attractor dynamics.

Software design

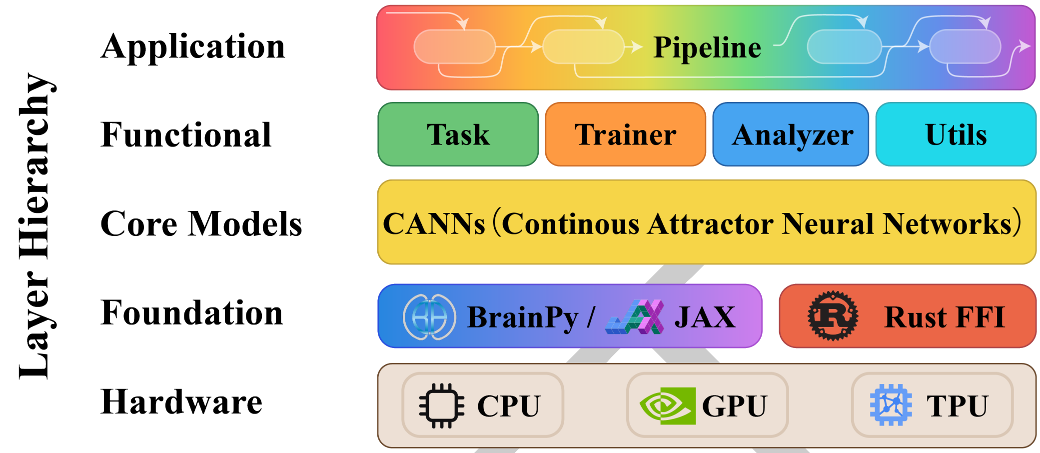

Figure 1: Layer hierarchy of the CANNs library showing five levels: Application (Pipeline orchestration), Functional (Task, Trainer, Analyzer, Utils modules), Core Models (CANN implementations), Foundation (BrainPy/JAX and Rust FFI backends), and Hardware (CPU/GPU/TPU support).

The CANNs library follows a modular architecture (Figure 1) guided by two core principles: **separation of concerns** and **extensibility through base classes**. The design separates functional responsibilities into five independent modules: (1) **Models** (`canns.models`) define neural network dynamics; (2) **Tasks** (`canns.task`) generate experimental paradigms and input data; (3) **Analyzers** (`canns.analyzer`) provide visualization and analysis tools; (4) **Trainers** (`canns.trainer`) implement learning rules for brain-inspired models; and (5) **Pipeline** (`canns.pipeline`) orchestrates complete experimental workflows.

Each module focuses on a single responsibility—models don't generate input data, tasks don't analyze results, and analyzers don't modify parameters. This separation ensures maintainability, testability, and extensibility. All major components inherit from abstract base classes (`BasicModel`, `BrainInspiredModel`, `Trainer`) that define standard interfaces, enabling users to create custom implementations that seamlessly integrate with the built-in ecosystem.

The library supports four distinct research workflows: (1) CANN modeling and simulation for studying attractor dynamics; (2) data analysis for processing experimental neural recordings; (3) brain-inspired learning with biologically plausible plasticity rules; and (4) end-to-end pipelines for automated parameter sweeps and reproducible experiments. All models inherit from BrainPy's `DynamicalSystem` base class, leveraging JAX's JIT compilation for GPU/TPU acceleration while maintaining simple Python APIs. For operations where Python overhead is significant, the companion `canns-lib` Rust library provides optional accelerated backends that can substantially improve performance in internal benchmarks for spatial navigation tasks and topological data analysis, without requiring code structure changes.

State of the field

While general-purpose neural network simulators like NEST (Gewaltig & Diesmann, 2007), Brian 2 (Stimberg et al., 2019), and NEURON (Hines & Carnevale, 1997) exist, they lack specialized support for continuous attractor networks. Existing CANN implementations remain fragmented, lab-specific codebases without standardized APIs or comprehensive tooling.

CANNs builds upon BrainPy (Wang et al., 2023, 2025), a powerful brain dynamics framework leveraging JAX (Bradbury et al., 2018) for JIT compilation and GPU/TPU acceleration. CANNs extends BrainPy with CANN-specific abstractions: standardized model implementations, task-generation APIs, analysis pipelines, and optional Rust-accelerated backends for performance-critical operations.

Research impact statement

CANNs provides full-detail, runnable modeling tutorials that reproduce recent CANN-related studies, packaged as standardized pipelines with consistent inputs, analysis steps, and visualizations. This allows researchers to verify published results and compare alternative mechanisms under a shared experimental setup, reducing the need for lab-specific reimplementations. The reproduced workflows include spike-frequency adaptation (Li et al., 2025; Mi et al., 2014) and theta-sweep dynamics in head-direction/grid and place-cell systems (Chu et al., 2024; Ji, Lomi, et al., 2025; Ji, Chu, et al., 2025), offering trusted baselines for new modeling and benchmarking studies.

AI usage disclosure

AI-assisted tools were used for code quality reviews and documentation writing. All core library code was written by human developers, and AI-generated content was reviewed and validated by the authors.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (no. T2421004 to S.W.),the National Key Research and Development Program of China (2024YFF1206500), the Science and Technology Innovation 2030-Brain Science and Brain-inspired Intelligence Project (no. 2021ZD0200204, S.W.).

References

Amari, S. (1977). Dynamics of pattern formation in lateral-inhibition type neural fields. *Biological Cybernetics*, *27*(2), 77–87. https://doi.org/10.1007/BF00337259

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable transformations of Python+NumPy programs* (Version 0.3.13). http://github.com/jax-ml/jax

Burak, Y., & Fiete, I. R. (2009). Accurate path integration in continuous attractor network models of grid cells. *PLoS Computational Biology*, *5*(2), e1000291.

Chu, T., Ji, Z., Zuo, J., Mi, Y., Zhang, W., Huang, T., Bush, D., Burgess, N., & Wu, S. (2024). Firing rate adaptation affords place cell theta sweeps, phase precession, and procession. *Elife*, *12*, RP87055. https://doi.org/10.7554/eLife.87055.4

Gewaltig, M.-O., & Diesmann, M. (2007). NEST (NEural simulation tool). *Scholarpedia*, *2*(4), 1430.

Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., & Moser, E. I. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature*, *436*(7052), 801–806. https://doi.org/10.1038/nature03721

Hines, M. L., & Carnevale, N. T. (1997). The NEURON simulation environment. *Neural Computation*, *9*(6), 1179–1209.

Ji, Z., Chu, T., Wu, S., & Burgess, N. (2025). A systems model of alternating theta sweeps via firing rate adaptation. *Current Biology*, *35*(4), 709–722. https://doi.org/10.1016/j.cub.2024.08.059

Ji, Z., Lomi, E., Jeffery, K., Mitchell, A. S., & Burgess, N. (2025). Phase precession relative to turning angle in theta-modulated head direction cells. *Hippocampus*, *35*(2), e70008. https://doi.org/10.1002/hipo.70008

Li, Y., Chu, T., & Wu, S. (2025). Dynamics of continuous attractor neural networks with spike frequency adaptation. *Neural Computation*, *37*(6), 1057–1101. https://doi.org/10.1162/neco_a_01757

Mi, Y., Fung, C., Wong, K., & Wu, S. (2014). Spike frequency adaptation implements anticipative tracking in continuous attractor neural networks. *Advances in Neural Information Processing Systems*, *27*.

O'Keefe, J., & Dostrovsky, J. (1971). The hippocampus as a spatial map: Preliminary evidence from unit activity in the freely-moving rat. *Brain Research*. https://doi.org/10.1016/0006-8993(71)90358-1

Stimberg, M., Brette, R., & Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator. *Elife*, *8*, e47314.

Taube, J. S., Muller, R. U., & Ranck, J. B. (1990). Head-direction cells recorded from the postsubiculum in freely moving rats. I. Description and quantitative analysis. *Journal of Neuroscience*, *10*(2), 420–435. https://doi.org/10.1523/JNEUROSCI.10-02-00420.1990

Wang, C., He, S., Luo, S., Huan, Y., & Wu, S. (2025). Integrating physical units into high-performance AI-driven scientific computing. *Nature Communications*, *16*(1), 3609.

Wang, C., Zhang, T., Chen, X., He, S., Li, S., & Wu, S. (2023). BrainPy, a flexible, integrative, efficient, and extensible framework for general-purpose brain dynamics programming. *eLife*, *12*, e86365. https://doi.org/10.7554/eLife.86365

Wu, S., Hamaguchi, K., & Amari, S. (2008). Dynamics and computation of continuous attractors. *Neural Computation*, *20*(4), 994–1025. https://doi.org/10.1162/neco.2008.10-06-378