The Journal of Open Source Software

DOI:

Software

- Review
- Repository
- Archive

Submitted:
Published:

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 serves 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 (**???**), entorhinal grid cells (**???**), and head direction systems (**???**). 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 addresses this gap by providing a unified Python toolkit built on a user-friendly and efficient programming framework BrainPy (**???**). It delivers: (1) standardized implementations of CANNs and related brain-inspired models, including mathematically tractable CANN models (**???**; **???**), adaptation-augmented CANN models (**???**; **???**), grid cell networks (**???**), 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

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

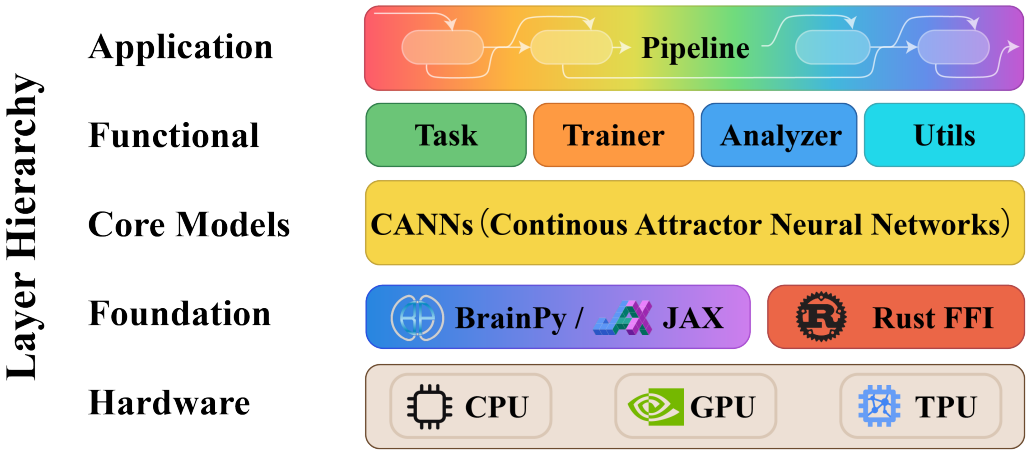

Figure 1: Layer hierarchy of the CANNs library showing five levels: Application (Pipeline orchestration), Functional (Task, Trainer, Analyzer, Utils modules), Core Models (CANN implementations), Foundation (BrainPy/JAX and Rust FFI backends), and Hardware (CPU/GPU/TPU support).

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—notably achieving 400× speedup for spatial navigation tasks and 1.13-1.82× speedup for topological data analysis—without requiring code structure changes.

Related Works

While general-purpose neural network simulators like NEST (**???**), Brian 2 (**???**), and NEURON (**???**) 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 (**???**), a powerful brain dynamics framework leveraging JAX (**???**) 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.

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

We acknowledge the BrainPy development team for providing the foundational framework upon which this library is built, and the broader open-source community for tools and libraries that enabled this work.

References