HELIOX: A GPU-Native Framework for Simulation and Training of Biophysically Detailed Networks

Abstract

Biophysically detailed neural networks represent a promising frontier for brain-inspired AI, offering intrinsic spatio-temporal dynamics to enhance the expressivity and computational density of deep learning systems. However, general-purpose deep learning frameworks suffer from a fundamental mismatch between their dense parallel optimizations and the irregular, tree-structured complexity of biological mechanisms. In this work, we propose **HelioX**, a **GPU-native** framework designed to unify high-performance simulation with scalable training. Unlike approaches that adapt biology to existing deep learning tools, HelioX adopts a "GPU-to-Biophysics" paradigm. We tailor the underlying GPU parallelism to biological structures by implementing custom-fused CUDA kernels for both the Dendritic Hierarchical Scheduling (DHS) algorithm and its gradient propagation. This design eliminates the runtime overhead of generic automatic differentiation and enables multi-stream concurrency for spike generation and equation assembly. Experimental results demonstrate that HelioX outperforms standard simulators (NEURON) by orders of magnitude and surpasses prior GPU-based solvers in both speed and scalability. We successfully train deep biophysical MLPs and organism-scale biophysical neural networks (e.g., the BAAIWorm C. elegans model) on a single consumer-grade GPU. HelioX establishes a new standard for computational efficiency, enabling the training of biophysically detailed models at scales previously unattainable.

1. Introduction

The remarkable prosperity of modern artificial intelligence is rooted in a foundational act of elegant simplification: the point-neuron abstraction. By distilling the intricate complexity of biological brains into scalar integration and nonlinear activations, this paradigm catalyzed the era of big data and large-scale model training. However, this abstraction discards a fundamental property of biological intelligence: spatiotemporal dynamics. Recent advances like Neural ODEs (Chen et al., 2018) and Liquid Time-Constant Networks (Hasani et al., 2021) demonstrate that reintroducing continuous-time dynamics yields superior robustness and data efficiency, suggesting that shifting from "instantaneous" inference to dynamic systems is a key frontier for AI.

If these models represent the first steps toward dynamic AI, biophysically detailed multi-compartment models (hereafter referred to as **detailed neurons**) represent its logical zenith. Unlike point-like dynamic systems, detailed neurons reintroduce spatial morphology as a primary computational dimension. Evidence suggests that this structural complexity is not mere biological redundancy but a source of immense power: a single detailed neuron can function as a 5-to-8 layer deep neural network (Beniaguev et al., 2021), natively executing sophisticated operations such as XOR logic and coincidence detection through the intrinsic interplay of its dendritic branches (Gidon et al., 2020). By coupling non-linear dynamics with physical morphology, detailed neurons achieve a level of computational density that standard paradigms cannot replicate, positioning them as the canonical candidate for next-generation, high-expressivity Artificial Neural Networks.

However, the transition from point-neurons to detailed biophysical models is bottlenecked by the lack of a scalable training framework. Unlike the inference of standard ANNs, which benefits from highly parallelizable matrix operations, detailed neuron models' inference is a fine-grained simulation process requiring the solution of coupled ODEs over time. Existing high-performance simulators designed for GPU parallelism often remain non-differentiable, while modern differentiable frameworks struggle with the topological sparsity of dendritic trees. Because these frameworks wrap biophysical equations into general-purpose engines designed for regular ANNs, they suffer from prohibitive memory overhead and computational redundancy, rendering the training of large-scale, morphologically detailed networks intractable.

In response to these challenges, we develop HELIOX, a GPU-native engine specifically architected to unify high-performance simulation with efficient differentiable training for biophysically detailed networks (BDNs). Impor-

tantly, HELIOX is designed to complement NEURON's modeling ecosystem. Models are built in NEURON using existing workflows, while HELIOX serves as a NEURON-compatible, GPU-native backend that accelerates simulation and enables efficient differentiable training.

HELIOX integrates a simulation core and a gradient computation unit within a cohesive, high-throughput framework. On the simulation side, HELIOX provides a native implementation of the DHS, further optimized through operator reduction and stream concurrency to maximize hardware utilization and memory efficiency. The training module features a dedicated gradient module that directly interfaces with the simulation core to track state variables. This design enables precise backpropagation across diverse neuronal morphologies, leveraging optimized gradient algorithms ((Zhang et al., 2023; Zhao et al., 2024)) to support large-scale network training. To facilitate accessibility, HELIOX offers a high-level modular API, allowing researchers to construct multi-layer biophysical architectures with ease. Our contributions are summarized as follows:

(a) GPU Native Simulation Engine: We develop a GPU-native engine based on the DHS method. By implementing this theoretically optimal parallelization strategy through custom CUDA kernels, we maximize hardware utilization and achieve unprecedented simulation speed for complex dendritic structures.

(b) Effective Training Module: We implement an efficient training module that provides GPU-native support for specialized gradient algorithms of detailed neurons. This framework includes a modular Python API for constructing biophysical MLPs, facilitating the scalable training of large-scale biophysical circuits.

(c)System Validation and Benchmarking: We demonstrate the efficacy of HelioX across diverse biophysical learning tasks. On the MNIST(LeCun et al., 2002) benchmark, we successfully train 3 to 5 layer biophysical MLPs that maintain stable accuracy while delivering significant gains in speed and memory efficiency. Furthermore, on the BAAIWorm *C. elegans* circuit-fitting task(Zhao et al., 2024), HelioX substantially accelerates the optimization loop and reduces peak GPU memory from 32GB to ∼6GB, enabling whole-circuit training on a single consumer GPU.

Why existing approaches fail. Current systems struggle to make *training* of large morphologically detailed networks practical because several bottlenecks compound in the inner loop:

- **Irregular tree solves do not map to dense-tensor kernels:** the cable equation induces a tree-structured sparse solve whose dependency pattern breaks standard GEMM-centric optimizations.
- **High overhead from fine-grained mechanism execution:** heterogeneous mechanisms lead to many small GPU kernels and frequent launch/synchronization overhead, especially when within-step ordering must be preserved.
- **Host-mediated event handling causes synchronization:** sparse spike delivery is latency-sensitive; naive CPU involvement introduces repeated host–device round trips and stalls.
- **AD-based training is memory/compute heavy:** differentiating through long-horizon ODE integration creates large computation graphs and state tapes, quickly becoming prohibitive for multi-compartment networks.

2. Background

From inference to simulation. Unlike standard artificial neural networks where *inference* is a sequence of feed-forward matrix operations, biophysically detailed neural networks (BDNs) are governed by continuous-time biophysical dynamics. Given time-varying inputs and initial states, the forward pass of a BDN is therefore a *numerical simulation* process: it advances the neuronal state over time according to biophysical differential equations. In this paper, we use **simulation** to denote the forward computation of a BDN.

Single and multi-compartment dynamics. BDNs are governed by continuous-time biophysical dynamics. After spatial discretization, a neuron is represented as a tree of coupled compartments. Let $V_i(t)$ be the membrane potential of compartment i with capacitance C_i. A compact form of the discretized cable equation is

$$C_i \frac{dV_i(t)}{dt} + I_{\mathrm{mem},i}(t) = I_{\mathrm{ext},i}(t) + \sum_{j \in \mathcal{N}(i)} I_{i \leftarrow j}(t), \tag{1}$$

where $\mathcal{N}(i)$ denotes axially connected neighbors and $g_{ij} \triangleq 1/R_{ij}$ is the axial conductance. We denote the axial current from compartment j to i as $I_{i \leftarrow j}(t) \triangleq g_{ij}\big(V_j(t) - V_i(t)\big)$. The transmembrane current aggregates biophysical mechanisms,

$$I_{\mathrm{mem},i}(t) \triangleq I_{\mathrm{ion},i}\big(V_i(t), \mathbf{g}_i(t); \boldsymbol{\theta}_{\mathrm{ion}}\big) + I_{\mathrm{syn},i}(t) + I_{\mathrm{leak},i}\big(V_i(t)\big), \tag{2}$$

and gating states typically follow additional ODEs,

$$\frac{d\mathbf{g}_i(t)}{dt} = \Phi\big(\mathbf{g}_i(t), V_i(t); \boldsymbol{\theta}_{\mathrm{ion}}\big). \tag{3}$$

Simulation pipeline. In a biophysically detailed simulator, the forward computation is a fixed-step time integration that advances compartment voltages and mechanism states.

We implement the model as a collection of **Mech** modules (e.g., ionic channel kinetics, synaptic dynamics, passive leak, and externally injected currents such as `IClamp`), each contributing transmembrane currents and/or local state updates.

A single simulation step can be viewed as a structured pipeline: (i) *event-driven network coupling*, which propagates spikes and applies synaptic effects (with delays) to update synaptic states; (ii) *mechanism evaluation*, which updates local states and accumulates transmembrane currents (including the synaptic currents determined by the delivered events); and (iii) *cable equation assembly and solve*, which advances voltages by solving the tree-structured coupling induced by cable theory.

Challenges for GPU-native simulation. This pipeline poses several challenges for GPU-native simulation: (1) the cable equation yields an irregular tree-structured solve that is poorly matched to dense-tensor optimizations; (2) heterogeneous mechanisms interact through shared states, requiring strict within-step ordering and explicit dependency enforcement under multi-stream execution; and (3) spike propagation is sparse yet latency-sensitive, and naive host-mediated handling can introduce frequent host-device synchronization.

Learning beyond automatic differentiation. Automatic differentiation (AD) enables end-to-end training of dynamical models by differentiating through the numerical solver. Recent work such as JAXLEY demonstrates that AD-based pipelines can support learning in multi-compartment neurons. However, backpropagating through the full ODE integration procedure can incur substantial computational and memory overhead due to long time horizons and large state spaces. An alternative direction is to derive analytical, structure-aware gradients that exploit the biophysical model structure.

DeepDendrite (Zhang et al., 2023) proposes an analytical synaptic gradient for training biophysically detailed networks. A common choice of forward readout is a time-averaged somatic voltage over a learning window,

$$\bar{v}_j \triangleq \frac{1}{t_e - t_s} \int_{t_s}^{t_e} v_j(t)\, dt, \tag{4}$$

from which task-specific predictions are computed and a loss E is evaluated. The key quantity for learning is the local error signal (gradient)

$$\delta_j \triangleq \frac{\partial E}{\partial \bar{v}_j}, \tag{5}$$

which can be computed at the output layer and propagated to hidden layers. Given δ_j, the per-timestep synaptic update for the k-th synapse from neuron i to neuron j is

$$\Delta W_{ijk}^{(n)} = \delta_j\, r_{ijk}\, g_{ijk}\, f\big(v_i(t_n)\big), \quad t_n \triangleq t_s + n\, dt, \tag{6}$$

where g_{ijk} is the synaptic conductance and r_{ijk} is the transfer resistance from the dendritic compartment hosting synapse $(i \to j, k)$ to the soma of neuron j. This morphology-dependent r_{ijk} makes the rule applicable to general multi-compartment neurons; a single-compartment neuron is a special case where r_{ijk} reduces to the compartment input resistance. Finally, weights are updated by accumulating $\Delta W_{ijk}^{(n)}$ over a learning window,

$$W_{ijk} \leftarrow W_{ijk} - \eta\, \frac{dt}{t_e - t_s} \sum_{n=0}^{N-1} \Delta W_{ijk}^{(n)}, \quad N \triangleq \frac{t_e - t_s}{dt}. \tag{7}$$

Transfer and input resistances can be computed by a reference simulator (e.g., NEURON).

Challenges for efficient learning systems. While analytical gradients provide an efficient learning rule, realizing high training throughput requires careful system design: (1) neuronal morphologies are heterogeneous and their states are distributed sparsely across dendritic trees rather than stored in regular dense tensors, often leading to fragmented, fine-grained operators and low GPU utilization; (2) different learning rules and neuron types require accessing biophysical quantities at different locations (e.g., compartment voltages, synaptic conductances, transfer resistances), demanding flexible indexing and efficient gathering/scattering of irregular data; and (3) training interleaves forward simulation with gradient computation and parameter updates, which requires low-overhead data exchange and synchronization between the simulation core and the learning modules.

3. The HelioX Framework

3.1. GPU-Native Simulation Framework

HelioX is *GPU-native by design*: we treat the GPU as the primary execution backend rather than an optional accelerator. Accordingly, we co-design data layouts, kernel boundaries, and scheduling around GPU constraints, and implement a set of targeted optimizations that map the simulator's irregular structure onto efficient GPU execution.

Concretely, we organize our GPU-native simulation stack into four components that correspond to the optimizations discussed below: (1) **Spike Processing** for sparse event handling, (2) **Unified Mechanism Template and Variable Registry** (MechTemp/VarStruct) for consistent mechanism instantiation and stable state access, (3) **Concurrent ODE Construction** for multi-stream mechanism evaluation and assembly, and (4) **Parallel ODE Solving** for the tree-structured cable solve.

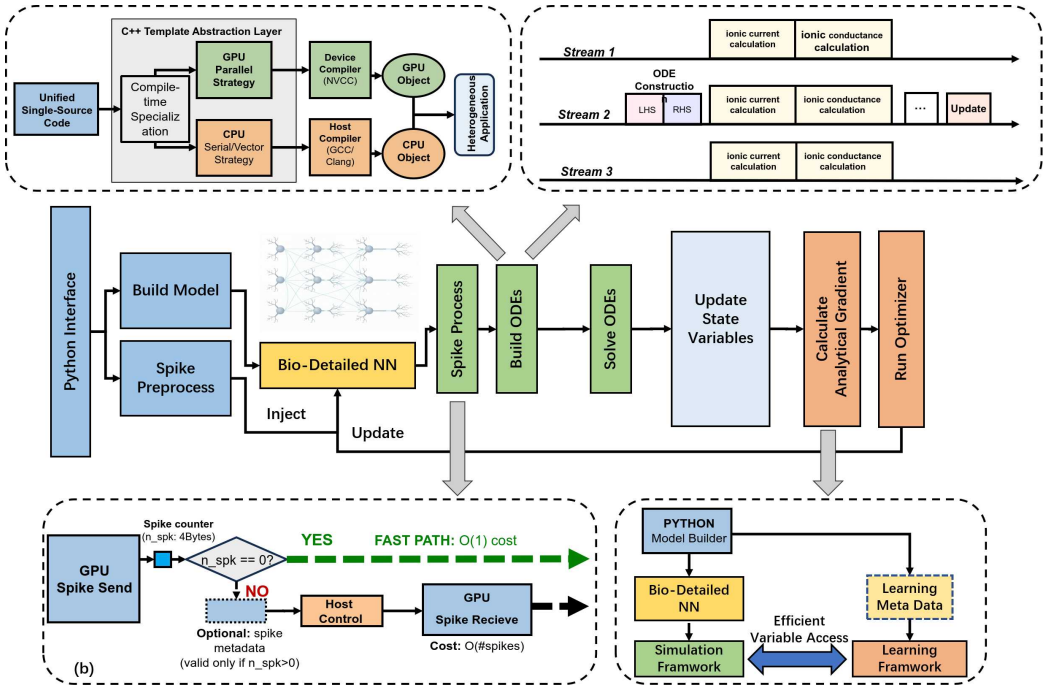

Figure 1. **Overview of HelioX.** HelioX bridges NEURON model specification and GPU-native execution. The runtime integrates (1) sparse spike processing, (2) a unified mechanism template and variable registry (MechTemp/VarStruct), (3) multi-stream concurrent ODE construction, (4) DHS-based parallel tree solving for the cable equation, and (5) an analytical-gradient learning module that directly binds to simulator states for efficient training.

Spike Processing. Biophysical events (spikes) are characterized by high temporal sparsity, often occurring in less than 1% of the simulation steps. Traditional simulators suffer from frequent host-device synchronizations to detect these rare events. To exploit this, we implement a two-stage event pipeline that replaces unconditional data transfers with a 4-byte gating mechanism. In each timestep, a lightweight kernel performs spike detection and returns only a single integer (spike count) to the host via pinned memory. If the count is zero, the framework bypasses all subsequent heavy operations, including the back-copy of spike indices and the invocation of post-synaptic delivery kernels. This design reduces the per-step overhead to $O(1)$ on spike-free steps, effectively eliminating redundant PCIe traffic and control-flow overhead in sparse networks.

Unified mechanism template and variable registry. HelioX introduces a template-based mechanism abstraction (MechTemp) to avoid backend divergence across CPU and GPU. In contrast to systems where CPU and GPU acceleration paths evolve with separate backends and code-generation toolchains (e.g., NEURON vs. CoreNEURON), each biophysical mechanism in HelioX is specified once as a node-level update and automatically instantiated into CPU and GPU kernels with consistent within-step semantics (differences are limited to floating-point rounding). Mechanism parameters and state variables are stored in a unified structured layout (VarStruct) and accessed through a uniform `var(var_name)` interface, enabling consistent addressing and memory interpretation across CPU/GPU.

In contrast, JAX/XLA-based frameworks (e.g., JAXLEY) abstract away device buffers and may reorder/reshape memory via compiler passes, which complicates enforcing a fixed layout and directly binding learning code to simulator states.

HelioX therefore exposes a reflective *variable registry* with stable identifiers that let external learning code (e.g., Python) locate and read/write simulator variables without ad-hoc name-to-index glue. This registry acts as a lightweight ABI between simulation and learning, improving interoperability and scalability.

Concurrent ODE construction. We decompose each fixed-step timestep into a set of GPU tasks (e.g., spike delivery, mechanism current evaluation, RHS/diagonal assembly, mechanism state updates, and recording/`VecPlay`) with explicit data dependencies . We exploit **multi-stream concurrency** to run independent mechanism kernels in parallel, and enforce NEURON-equivalent within-step semantics using **CUDA events** (`cudaStreamWaitEvent`) instead of device-wide synchronization. This event-driven scheduling reduces CPU synchronization overhead and improves GPU

utilization when mechanism kernels are small and heterogeneous. To further reduce GPU kernel launch overhead, we apply lightweight kernel fusion to cut launch overhead and redundant memory traffic.

Parallel ODE Solving. Solving the multi-compartment cable equation reduces to a sparse linear solve on a tree, typically performed by the Hines method. This step is challenging to parallelize because the elimination and back-substitution must follow a strict dependency order induced by the dendritic tree; naive parallel execution can violate these dependencies and thus produce incorrect results.

To address this, we adopt the **Dendritic Hierarchical Scheduling (DHS)** algorithm from DeepDendrite (Zhang et al., 2023). DHS organizes the tree solve into dependency-respecting levels, exposing maximal parallelism while preserving the exact Hines semantics. DeepDendrite proves DHS to be theoretically optimal in terms of parallel time for fine-grained neuron (multi-compartment) Hines solves, making it a principled foundation for GPU-native ODE solving.

3.2. Learning Framework

HelioX adopts an *analytical-gradient* learning approach rather than relying on generic automatic differentiation (AD).

By leveraging structure-aware gradient derivations for biophysical models, analytical gradients avoid mechanically backpropagating through long-horizon ODE integration, and thus substantially reduce both compute and memory overhead.

This advantage is critical for morphologically detailed neurons: a single cell may contain hundreds of compartments and heterogeneous mechanisms, and a network forward pass typically corresponds to a continuous simulation trajectory (e.g., tens to thousands of milliseconds) discretized into many timesteps. As a result, training produces large volumes of irregular, time-indexed biophysical signals (voltages, conductances, and mechanism states), making efficient *binding, indexing, aggregation, and update* of such data a primary systems challenge.

Modeling–execution workflow and freeze point. HelioX is designed to be compatible with the established NEURON modeling ecosystem. Model specification—including morphology parsing, mechanism insertion, connectivity construction, and event routing—is performed in NEURON.

After model construction, HelioX imports the model into its GPU-native runtime and enters an *execution phase* in which all learnable parameters and training signals are bound to a stable set of runtime identifiers. This *freeze point* decouples high-frequency training from frontend object semantics: the learning backend only consumes a compact metadata description (e.g., the set of trainable variables, optimizer configuration, and hyperparameters) and operates directly on the simulator's device-resident states.

Handle-based state binding and temporal buffers. To support learning on irregular dendritic trees, HelioX exposes a reflective variable registry that maps biophysical quantities to their underlying GPU buffers.

However, repeatedly resolving variables by name or metadata at every timestep would introduce non-negligible overhead in the tight training loop. Therefore, HelioX uses a *handle-based caching* mechanism: variable resolution is performed once and cached as stable handles (device pointers plus shape/stride metadata), enabling $O(1)$ access for subsequent reads/writes during training.

Different objectives require different temporal views of simulator states. For time-averaged readouts (e.g., voltage averages over a learning window), HelioX supports *in-simulation accumulation* so that reductions are performed online during simulation, reducing post-processing and data movement. For trajectory-level objectives that require time series, the runtime maintains a device-side temporal buffer (window/ring buffer) with contiguous layout, allowing learning kernels to consume sequential data efficiently without repeated fragmented gathering.

Two learning workflows: structured composition and general biophysical models. HelioX provides two complementary workflows to cover common research settings.

For structured feed-forward architectures (e.g., biophysical MLPs built from simplified or passive neuron models), HelioX offers a PyTorch-like `Sequential` API for composing multi-layer networks and a batched input-encoding interface (e.g., converting a vector input into rate-coded spike sequences—one per input dimension, to drive input neurons) to reduce Python-side overhead. For general biophysical circuits with heterogeneous morphologies, mechanisms, and recurrent connectivity, users build the model in NEURON and then invoke HelioX's learning runtime by passing the exported model along with the training configuration; the runtime binds trainable parameters via the variable registry and executes simulation, gradient computation, and parameter updates on the GPU.

4. Experimental Results

4.1. Numerical Fidelity and Scalability

We evaluate HelioX by comparing numerical fidelity and runtime efficiency against established simulators (NEURON, CoreNEURON, and DeepDendrite). All experiments

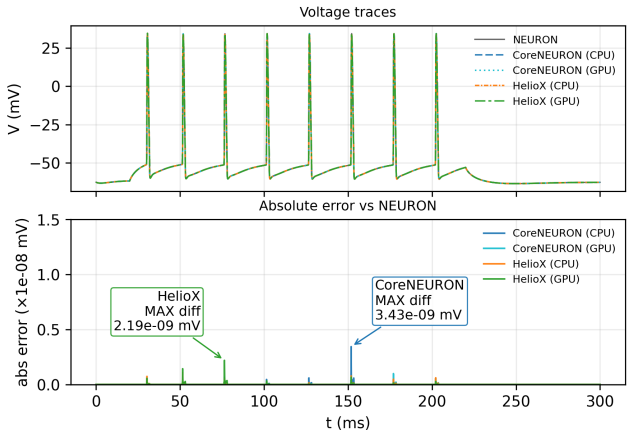

Figure 2. **Numerical fidelity of HelioX (L5PC).** We compare HelioX against NEURON on a biologically detailed Layer 5 pyramidal cell (L5PC). HelioX matches NEURON with maximum absolute error below 10^{-8} mV and reproduces identical spike timing.

run on an AMD Ryzen 9 9950X and an NVIDIA RTX 5090 (32GB), spanning a single detailed L5 pyramidal cell (L5PC) up to a 500-cell passive HPC network; HelioX uses FP64 throughout to match NEURON.

We measure accuracy by the maximum absolute error (MAE) of membrane potentials and spike-time consistency, using NEURON as reference. Figure 2 shows MAE below 10^{-8} mV (i.e., at floating-point round-off level), which is far smaller than voltage changes induced by typical integration step sizes; accordingly, in all our tests HelioX reproduces identical spike times as NEURON. Additional fidelity results for the passive 500-HPC benchmark are reported in Appendix A.

Having established this high degree of numerical correctness, we further examine the computational advantages of our framework. Computational efficiency is measured by the speedup ratio in wall-clock execution time relative to the baselines. Through our experiments over the models and frameworks mentioned above, we find that HelioX reaches a speedup of 1.61–2.25× over CoreNEURON and 1.69–2.60× over DeepDendrite, as summarized in Table 1. Especially, we also benchmark the 500-HPC network against JAXLEY, ensuring standardized experimental configurations for an equal comparison. In this test, HelioX also achieves a significant speedup of 3.12×. These results demonstrate that HelioX effectively combines the numerical fidelity of traditional CPU simulators with the superior throughput of modern GPU hardware.

4.2. Learning in Deep Biophysical Networks

Beyond simulation, we verify the learning capability, computational efficiency and scalability of the learning subsystem of HelioX by conducting an image classification task on the MNIST dataset (LeCun et al., 2002). Following the benchmark proposed in DeepDendrite (Zhang et al., 2023), this task serves as a comprehensive comparative study against three baseline frameworks: CoreNEURON, DeepDendrite, and JAXLEY.

Network Architecture and Task Setup. We evaluate HelioX on a *deep biophysical* multi-layer perceptron (MLP) for MNIST classification, following the DeepDendrite benchmark (Zhang et al., 2023) (a structured feed-forward instance of our BDN setting). The network replaces point neurons with multi-compartment Hippocampal Pyramidal Cells (HPCs): an input layer of 784 single-compartment cells (one per pixel) drives 1–3 hidden layers of 64–256 multi-compartment HPC neurons, followed by an output layer of 10 neurons.

Layers are fully connected by *graded synapses* from the presynaptic soma to a postsynaptic dendritic compartment; the trainable weight W_{ij} scales the synaptic strength. For classification, we compute a voltage-based readout by averaging the somatic membrane potential $v_j(t)$ of each output neuron over a learning window $[t_s, t_e]$ (Eq. (4)), then apply a Softmax classifier and optimize cross-entropy loss.

This setup requires HelioX to simulate large coupled ODE systems and compute analytical gradients efficiently across the full network topology.

Accuracy and Convergence. We first evaluate whether HelioX maintains learning integrity. HelioX reaches a test accuracy of 95.9% after 4 epochs. Due to the prohibitive computational cost of JAXLEY's training in our high-fidelity setting, we compare our performance with its originally reported accuracy of 94.2%. This parity validates that our optimization techniques do not introduce gradient drift or numerical instability.

Throughput and Memory. The computational superiority of HelioX on this *deep biophysical MLP* task is summarized by the throughput and memory numbers reported in Tables 3 and 2. Notably, HelioX outperforms JAXLEY by 11.94× on training and 4.33× on testing. Compared to DeepDendrite, HelioX achieves a 1.87× speedup on training and 2.46× on testing, which we attribute to our GPU-native analytical-gradient training together with multi-stream concurrency in the simulation/learning runtime. Strikingly, HelioX achieves a dramatic speedup of **43.67**× on training and **44.72**× on testing compared to the CoreNEURON baseline. All throughput/memory results are measured on the 3-layer network. Peak GPU memory usage is reported in Table 2. Compared to JAXLEY, HelioX only uses 0.51× of memory.

Table 1. Comprehensive performance comparison across simulation benchmarks. **HelioX** achieves up to $\sim 2.25\times$ speedup compared to the strongest baseline (CoreNEURON). The best result in each row is highlighted in **bold**. The speedup factor (in parentheses) is calculated relative to CoreNEURON. "-" indicates the method does not support the task or data is unavailable.

Benchmark Task	NEURON	DeepDendrite	JAXLEY	CoreNEURON	HelioX
Single L5PC	2.45	3.30	-	2.99	**1.33 (2.2×)**
L5PC Network (20 cells)	48.87	4.05	-	2.82	**1.56 (1.8×)**
Passive 500 HPC	131.48	5.61	10.30	5.34	**3.30 (1.6×)**
Active 500 HPC	134.17	9.55	-	8.04	**3.92 (2.1×)**

Table 2. Peak GPU memory usage on the MNIST MLP task (MiB).

Framework	Peak Memory (MiB)
CoreNEURON	1079
DeepDendrite	1321
JAXLEY	2219
HelioX (Ours)	**1121**

Table 3. Training and inference efficiency on the MLP task (time to train/test 1000 samples; batch size $= 4$). Dashes denote frameworks that cannot extend to those depths. Rows show training (top) and testing (bottom) times in **seconds**. Architectures: **3-layer**=784-64-10, **4-layer**=784-128-64-10, **5-layer**=784-256-64-10. Input/output layers use single-compartment cells; hidden layers use PassiveHPC cells.

Network	CoreNEURON	DeepDendrite	JAXLEY	HelioX (Ours)
3-layer	220.99 198.57	9.47 10.92	60.41 19.24	**5.06** **4.44**
4-layer	–	–	–	**7.09** **5.54**
5-layer	–	–	–	**12.32** **8.65**

Scalability. With HelioX's flexible `Sequential` API, we further evaluate scalability by training deeper *deep biophysical MLP* variants (4-layer and 5-layer); results are reported in Table 3. In contrast, DeepDendrite and JAXLEY rely on hard-coded scripts/topologies in their benchmark implementations, which prevents the same depth-scaling tests under matched settings.

4.3. Large-Scale Biophysical Network Training: *C. elegans* Whole-Brain Simulation

To evaluate HelioX on organism-scale learning, we replicated the biological data fitting task from BAAIWorm, a recently proposed whole-brain model of *C. elegans* (Zhao et al., 2024). This benchmark optimizes network parameters of a biologically detailed circuit so that its simulated activity matches experimental recordings.

The target circuit comprises 136 multi-compartment Hodgkin–Huxley neurons (sensory neurons, interneurons, and motor neurons) connected according to the *C. elegans* connectome (Zhao et al., 2024). Following BAAIWorm, we treat both chemical synapses and electrical gap junctions as trainable and optimize their strengths; the original work also considers additional trainable factors such as synapse polarity (excitatory vs. inhibitory) and external input currents to a subset of sensory neurons (Zhao et al., 2024).

Table 4. Training time (per epoch) and peak GPU memory on the BAAIWorm *C. elegans* circuit-fitting task (RTX 4090). Speedup is reported in parentheses, and memory savings are reported as percentage reduction (↓).

Method	Time / epoch (s)	Peak GPU mem. (GB)
CoreNeuron	14840	47.4
HelioX	5100 (2.91×)	5.2 (↓ 89.0%)
HelioX (DS=5)	60 (247.33×)	1.6 (↓ 96.6%)

Training target and loss. BAAIWorm defines the training target using whole-brain calcium imaging. Specifically, it computes the Pearson correlation matrix from experimentally recorded Ca^{2+} signals of 65 identified head neurons. We minimize the mean squared error (MSE) between the correlation matrix produced by the simulation and the experimental correlation matrix (Zhao et al., 2024).

Optimization algorithm. Following BAAIWorm, we adopt the optimization algorithm in (Zhao et al., 2024) for multi-compartment recurrent circuits. At a high level, it performs gradient-based optimization and uses mathematical approximations to compute *approximate gradients* for the recurrent network, which are then used to update trainable synaptic and coupling parameters. We adopt the same task definition and validate HelioX by matching the training trajectory against a CoreNEURON-based baseline.

Learning-time downsampling. Detailed-neuron simulation typically requires a fine integration step (e.g., 0.5 ms), while many learning signals evolve smoothly at this temporal resolution. HelioX therefore supports *learning-time downsampling* for gradient computation: we compute gradients using a temporally downsampled subset of the simulated trajectory (while keeping the forward simulation at the original step size). This reduces the effective history length and the number of gradient evaluations, lowering compute

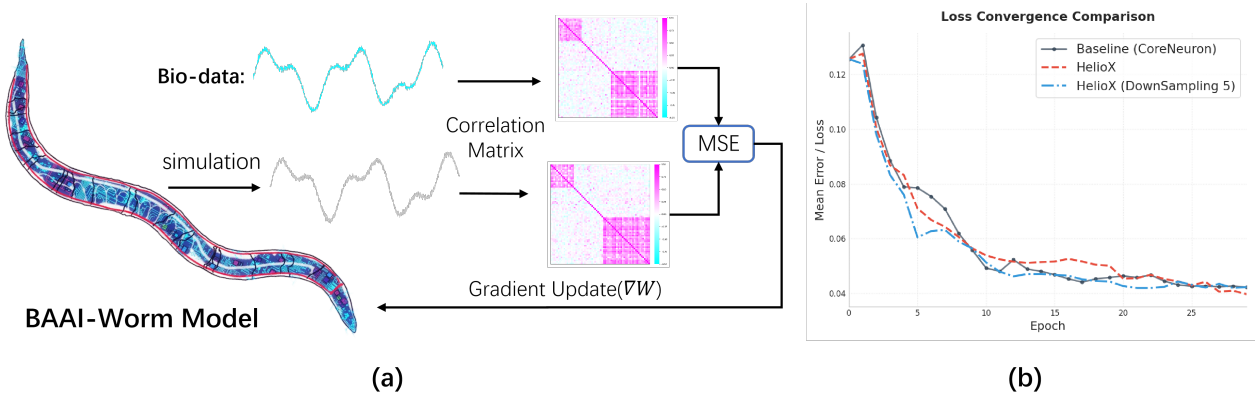

Figure 3. **Training the BAAIWorm whole-brain model with HelioX. (a)** Schematic of the biological data fitting task: optimizing chemical and electrical synaptic parameters of a 136-neuron biophysical circuit to match experimental whole-brain Ca^{2+} recordings (via correlation-matrix supervision). **(b)** Training loss curves. The trajectory of HelioX (solid line) aligns with the CoreNeuron-based baseline (dashed line), validating numerical correctness under the same task definition.

and memory-bandwidth pressure.

We evaluate training throughput and peak memory usage on two hardware configurations: an NVIDIA RTX 4090 (48GB memory configuration), results are summarized in Table 4. With $5\times$ downsampling enabled, HelioX trains one epoch in about 60 seconds on an RTX 4090 while reaching the same optimization target (final loss) as the original setting, corresponding to a $\sim 247\times$ speedup over the 4.1-hour baseline.

HelioX substantially reduces peak memory usage ($\sim$5.2GB vs. 47GB for the baseline). These results indicate that HelioX significantly lowers the hardware barrier for full-brain biophysical training, enabling complex scientific tasks on standard consumer-grade GPUs.

5. Discussion

HelioX makes training and fitting of biophysically detailed networks practical by co-designing a GPU-native simulator and an analytical-gradient learning runtime, while remaining compatible with NEURON model specification. Across both structured deep biophysical MLPs and an organism-scale *C. elegans* circuit-fitting benchmark, HelioX improves throughput and memory efficiency, lowering the hardware barrier for scalable biophysical learning.

6. Conclusion

In this work, we presented HelioX, a GPU-native framework that unifies high-performance simulation with scalable learning for biophysically detailed neural networks. HelioX combines a GPU-oriented simulation stack (including sparse spike processing, a unified mechanism template with a stable variable registry, multi-stream concurrency, and DHS-based tree solves) with an analytical-gradient learning runtime that avoids the overhead of generic automatic differentiation. Across both structured deep biophysical networks and an organism-scale *C. elegans* circuit-fitting benchmark, our results demonstrate that HelioX substantially improves throughput and memory efficiency, lowering the practical barrier to training morphologically detailed networks.

More broadly, HelioX aims to make detailed neurons usable in two complementary roles: as expressive computational units for brain-inspired AI and as scalable, high-fidelity models for computational neuroscience. We hope this capability will enable both communities to iterate faster on increasingly realistic models and to explore how biological structure and dynamics shape learning and computation.

References

Beniaguev, D., Segev, I., and London, M. Single cortical neurons as deep artificial neural networks. *Neuron*, 109 (17):2727–2739, 2021.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Deistler, M., Kadhim, K. L., Pals, M., Beck, J., Huang, Z., Gloeckler, M., Lappalainen, J. K., Schröder, C., Berens, P., Gonçalves, P. J., and Macke, J. H. Differentiable simulation enables large-scale training of detailed biophysical models of neural dynamics. *bioRxiv*, 2025. doi: 10.1101/2024.08.21.608979.

Fang, W., Chen, Y., Ding, J., Yu, Z., Masquelier, T., Chen, D., Huang, L., Zhou, H., Li, G., and Tian, Y. Spikingjelly: An open-source machine learning infrastructure platform for spike-based intelligence. *Science Advances*, 9(40): eadi1480, 2023.

Gidon, A., Zolnik, T. A., Fidzinski, P., Bolduan, F., Papoutsi, A., Poirazi, P., Holtkamp, M., Vida, I., and Larkum, M. E. Dendritic action potentials and computation in human layer 2/3 cortical neurons. *Science*, 367(6473):83–87, 2020.

Hasani, R., Lechner, M., Amini, A., Rus, D., and Grosu, R. Liquid time-constant networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7657–7666, 2021.

Hines, M. Efficient computation of branched nerve equations. *International Journal of Bio-Medical Computing*, 15(1):69–76, 1984. doi: 10.1016/0020-7101(84)90008-4.

Hodgkin, A. L. and Huxley, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117 (4):500, 1952.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 2002.

London, M. and Häusser, M. Dendritic computation. *Annu. Rev. Neurosci.*, 28(1):503–532, 2005.

Rall, W. Branching dendritic trees and motoneuron membrane resistivity. *Experimental neurology*, 1(5):491–527, 1959.

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

Zhang, Y., He, G., et al. A gpu-based computational framework that bridges neuron simulation and artificial intelligence. *Nature Communications*, 14:5798, 2023. doi: 10.1038/s41467-023-41553-7.

Zhao, M., Wang, N., Jiang, X., Ma, X., Ma, H., He, G., Du, K., Ma, L., and Huang, T. An integrative data-driven model simulating c. elegans brain, body and environment interactions. *Nature Computational Science*, 4(12):978–990, 2024.

A. Additional Numerical Fidelity Results

Passive 500-HPC benchmark. For this benchmark, the most recent CoreNEURON release we tested (9.0.0) crashed when running the demo, so we report accuracy comparisons only between HelioX (GPU mode) and NEURON. Figure 4 shows that the observed voltage error remains within typical floating-point round-off noise.

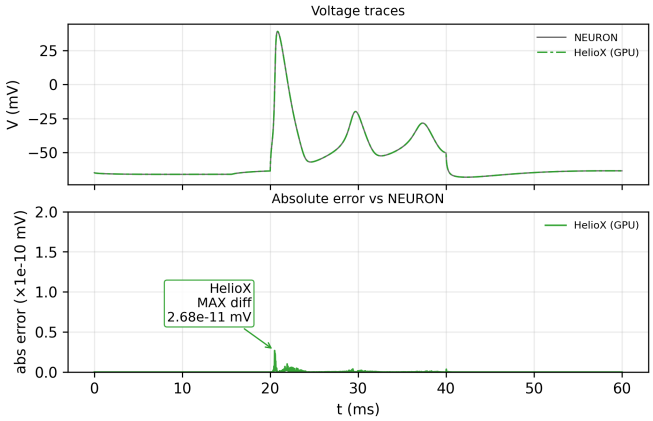

Figure 4. **Numerical fidelity of HelioX (Passive 500-HPC).** We compare HelioX against NEURON on the passive 500-HPC benchmark. The observed voltage error remains within typical floating-point round-off noise.

B. Related Work

B.1. Biophysically Detailed Models

While point neurons abstract cells into single nodes, detailed neurons explicitly incorporate morphology as a primary computational dimension. Grounded in Cable Theory (Rall, 1959), the dendritic tree is discretized into hierarchical compartments where membrane potentials evolve via coupled systems of ODEs. These systems are driven by transmembrane ion channel kinetics (Hodgkin & Huxley, 1952). This morphological complexity translates into powerful structural inductive biases. Rather than acting as passive cables, dendrites function as active nonlinear processors (London & Häusser, 2005) that provide a biophysical substrate for high dimensional computation. Mathematically, this structure has been linked to deep network expressivity (Beniaguev et al., 2021). This enables individual neurons to natively execute complex operations such as XOR logic and direction selective filtering (Gidon et al., 2020), functions that would otherwise require multilayer ANNs.

B.2. Simulation and Training Frameworks

Although traditional simulators such as NEURON and CoreNEURON (Hines, 1984) remain the gold standard for accuracy, their lack of native automatic differentiation makes them incompatible with modern gradient-based pipelines. DeepDendrite (Zhang et al., 2023) introduced the Dendritic Hierarchical Scheduling (DHS, (Zhang et al., 2023)), a theoretically optimal method, to maximize GPU parallelism for simulation, yet it lacks a differentiable wrapper, leaving a gap between high-performance inference and end-to-end training. Learning frameworks such as SpikingJelly(Fang et al., 2023) effectively integrate spiking neural networks with PyTorch but are tailored for simple leaky integrate and fire models. They do not support the hierarchical compartment topologies or complex systems of equations required for detailed morphological modeling. Existing differentiable biophysical frameworks such as BrainPy (Wang et al., 2023) and JAXLEY (Deistler et al., 2025) represent the state of the art by leveraging automatic differentiation engines like JAX. These platforms adopt a generic mapping approach that translates biophysical equations into standard computational primitives. Without domain specific sparse solvers, the computation graph for dendritic trees becomes prohibitively large during backpropagation. This leads to severe memory overhead and limits scalability to small circuits.