SoftwareX 34 (2026) 102648

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

OnlineNeuro: Active online learning for effective exploration of neural simulation parameters

Diego Nieves Avendano[a,*], Arne Callaert[a], Philipp Schnepel[b], Vojkan Mihajlovic[b], Femke Ongenae[a], Sofie Van Hoecke[a]

[a] IDLab, Ghent University - imec, Gent, Belgium
[b] imec Netherlands, Eindhoven, Netherlands

ARTICLE INFO

Keywords:
Neuroscience simulation
Neurostimulation
Active learning
Online optimization
Gaussian process
Deep learning

ABSTRACT

Neuromodulation studies require efficient exploration of high-dimensional stimulation spaces, where heuristic tuning is often slow and suboptimal. We present OnlineNeuro, an open-source Python framework that combines active learning with neural simulators (AxonSim, Cajal, and AxonML). The package offers a unified interface for experiment setup, model training, adaptive sampling, and reporting. By prioritizing informative queries, OnlineNeuro improves sample efficiency for parameter exploration and meta-model construction. We demonstrate the framework on neural simulation use cases and benchmark tasks.

Metadata

Code metadata (mandatory).

Nr.	Code metadata description	Please fill in this column
C1	Current code version	1.0
C2	Permanent link to code/repository used for this code version	https://github.com/predict-idlab/OnlineNeuro
C3	Permanent link to Reproducible Capsule	https://codeocean.com/capsule/8662395/tree
C4	Legal Code License	imec License.
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python, MATLAB
C7	Compilation requirements, operating environments & dependencies	
C8	If available Link to developer documentation/manual	
C9	Support email for questions	diego.nievesavendano@ugent.be

1. Motivation and significance

Neuromodulation is a class of therapeutic techniques in which electrical stimuli are applied to neural tissue to activate, suppress, or regulate physiological function [8]. Treatment performance depends on many coupled stimulation parameters (e.g., amplitude, frequency, and waveform shape). Their Cartesian combinations define large search spaces, and this complexity grows quickly with dimensionality [4,5]. In practice, this makes exhaustive exploration costly or infeasible.

Clinical and preclinical stimulation routines are therefore often tuned with heuristic, expert-driven workflows. While practical, these workflows can require many evaluations and may converge to suboptimal settings, particularly when responses vary across patients and over

* Corresponding author.
Email address: diego.nievesavendano@ugent.be (D. Nieves Avendano).

https://doi.org/10.1016/j.softx.2026.102648
Received 19 December 2025; Received in revised form 27 March 2026; Accepted 7 April 2026
Available online 13 April 2026

time [2]. Anatomical variability, electrode placement differences, and temporal changes in physiology all contribute to this challenge.

Active learning addresses this bottleneck by selecting the next experiment using expected information gain, reducing evaluations while maintaining model quality or optimization progress [7,12,14,22]. Here, "online" refers to sequential model updating as new samples arrive, rather than real-time closed-loop therapy.

OnlineNeuro provides a unified framework to couple active-learning methods with neurostimulation simulators. It integrates TensorFlow, GPflow, and Trieste models with AxonSim, Cajal, and AxonML interfaces, enabling consistent experiment execution, logging, and analysis across tools [1,17,20].

Computational neuroscience simulators such as NEURON are commonly used in early-stage study design and hypothesis testing, but broad parameter sweeps can be computationally intensive in high-dimensional settings [11].

This paper is structured as follows: Section 2 summarizes active-learning concepts relevant to neurostimulation, Section 3 describes architecture and functionalities, Section 4 presents representative use cases, and Sections 5 and 6 discuss impact and conclusions.

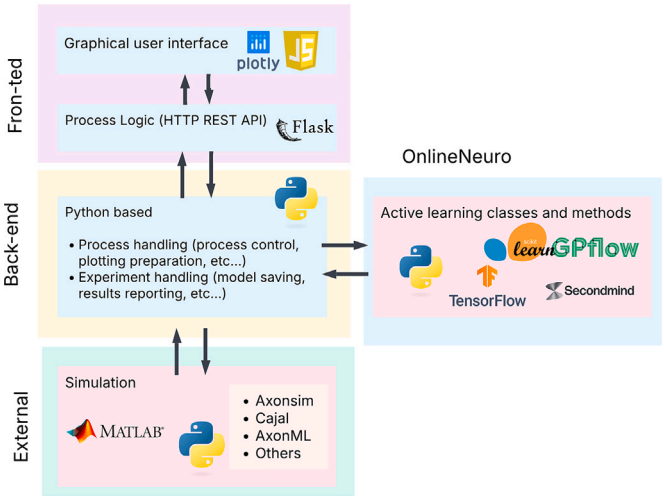

Fig. 1. OnlineNeuro architecture.

2. Active learning

Active learning is a machine-learning paradigm in which a model iteratively queries the most informative samples, instead of relying on large uniformly labeled datasets. In each iteration, a probabilistic learner estimates predictions and uncertainty, an acquisition function ranks candidate points, and an oracle (simulator or experiment) returns the label for selected points.

The key components are therefore: (i) the learner, which should provide calibrated predictive uncertainty; (ii) the query strategy, which determines which point(s) to evaluate next; and (iii) the oracle, which can be a computational model, a laboratory experiment, or an expert annotator. Depending on the objective, stopping criteria and evaluation metrics are used to decide when performance is sufficient.

In neurostimulation, labels are expensive because each query may require a simulation run or biological measurement. Sample-efficient querying is therefore critical, and common strategies include uncertainty sampling, improvement-based criteria, and hybrid exploration–exploitation rules [12,14]. OnlineNeuro exposes these components through interchangeable models and samplers, enabling reproducible comparisons of search strategies under fixed evaluation budgets.

3. Software description

OnlineNeuro is an open-source Python package for efficient evaluation of neurostimulation configurations through surrogate modeling with machine-learning models from multiple libraries.

The central principle of OnlineNeuro is seamless integration of simulators (e.g., AxonSim in MATLAB and AxonML in Python) with ML frameworks (e.g., TensorFlow and GPflow) [1,17]. The library also provides a graphical user interface (GUI) for real-time visualization, and its classes and methods can be extended for custom problems.

3.1. Software architecture

OnlineNeuro is developed with a three-layer structure composed of a front-end, a back-end, and the external interfaces. Fig. 1 presents the structure, subcomponents, communication routes, and associated technologies in each layer.

The layer functionality is divided as follows:

- **Front-end**. Divided into a Graphical User Interface (GUI) and a process logic. The GUI is written in JavaScript, HTML, and CSS for templating, and Plotly [21] for plotting. Its main function is to allow users to configure experiments without having to write Python and to provide online feedback/results of the experiment through logging and plots. The process logic components allow loading saved experiment results and models for further optimization as well as logic concerning correct problem parametrization (e.g., valid acquisition functions depending on the model's objective).
- **Back-end**: An HTTP REST API that mediates communication between the GUI the active learning layer and the simulators in a unified way.
- **Active learning**: This is the core Python component of OnlineNeuro and handles ML model initialization from different libraries (Trieste, GPflow and TensorFlow). It contains dedicated classes and methods for active learning designed to specifically address different neurostimulation use cases, and also extends the functionality of pre-existing components. This layer can be used independently of the GUI for custom optimization problems.
- **Simulation**: The simulators are packages developed by other parties for neural simulation. Currently, OnlineNeuro integrates with AxonSim (MATLAB), Cajal and AxonML (Python). Wrapper functions have been written to make it easier to interact with the simulators.

3.2. Software functionalities

OnlineNeuro contributions can be grouped into three components: (i) core GUI/backend modules for experiment setup, reporting, and model storage with limited coding effort; (ii) active-learning components that provide configurable access to advanced ML models and acquisition functions, extending Trieste-style workflows for a consistent user experience; and (iii) AxonSim extensions for custom temporal stimulation patterns and deeper control of simulator parameters than the default MATLAB GUI. Table 1 summarizes these contributions, while Table 2 maps added models/acquisition functions to use cases. Additional API details are available in the repository documentation; for a broader review of acquisition functions and Bayesian optimization, see Frazier [9].

4. Illustrative examples

OnlineNeuro examples are available through the GUI, Jupyter notebooks, and Python scripts. Here we present the GUI and two simulation use cases that highlight key functionalities.

Table 1
Features summary per layer in OnlineNeuro.

Component	Features
Front-end	Graphical user interface (GUI). Dynamic handling of model and problems constraints.
Back-end	Process logic and plotting preparation.
Online Neuro (active learning)	`MCDropout`: Monte Carlo Artificial Neural Network with uncertainty estimation via Dropout. `ProbabilisticNetwork`: Bayesian Neural Network for classification problems. `DiscreteBatchSampling`: Acquisition rule to perform batch sampling of arbitrary scoring functions. Multi-output regression scoring functions: Maximum variance sampling `TemporalVariance` and weighted variance temporal gradient `TemporalVarianceMovement`. `MaxCoverageSampler` and `DiversityUncertaintySampler` acquisition rules for low-discrepancy and weighted uncertainty/low-discrepancy sampling. `ExpectedImprovementXi`: Greedy/Explorative modification of ExpectedImprovement.
External: AxonSim	Extended experiment configuration for precise electrode configuration. Support for custom parametric functions and better control for time dependent functions. Signal post-processing routines.
Selected code examples (presented here)	Action potential meta-modeling in Cajal. Action potential meta-modeling in AxonSim. Diverse toy problems (regression, classification) for native Python and MATLAB.

4.1. Graphical user interface

The GUI simplifies experiment configuration, reporting, and result exploration. This is useful when multiple experiments are run with minimal code changes, for example when testing different waveforms, search spaces, or electrode counts. Traditionally, each of these tasks requires changing the underlying Python code. Using predefined JSON configuration files, the front-end exposes required inputs and the back-end adapts the execution logic accordingly. Furthermore, the GUI, shown in Fig. 2, allows visualization of results during data collection, which can guide users, especially during expensive simulations.

The GUI can be extended to cover new problems and use new models, by extending the interface in `/frontend/`, and the models and configurations in `/config/`. In case the users are interested in extending the plotting functionalities, they can extend the JS/Plotly functions in `/frontend/templates/`.

4.2. Action potential modeling with AxonSim

OnlineNeuro can be used to evaluate AxonSim (MATLAB) problems. In this example, we create a meta-model of the action potential (AP) over different stimulation parameters. The objective is to create a meta model of how these waves change over the search space, which in turn can be used to select parameters for an intended activation pattern. The reproducible code for this can be found in `notebooks/Axonsim_AP_modelling.ipynb`.

This example highlights OnlineNeuro's capability for multi-output modeling. The setup uses a shallow dense network as base model, Monte Carlo Dropout (MC-Dropout) for uncertainty estimation, and a variance-based acquisition rule. MC-Dropout is an ideal uncertainty estimator for problems where the regression output has a high dimensionality, which often cannot be handled by models such as GP or BNN due to high memory requirements [10]. The Temporal Variance–Movement acquisition weights predictive uncertainty and temporal gradients to improve estimation of action-potential peaks. More details of the acquisition function are provided in the supplementary material.

The simulations use the same axon configuration in all cases, and only the pulse configuration is modified. Table 3 shows the configuration of the experiment. The search space consists of two features, namely the current and pulse width of a double pulse. A double pulse consists of two monophasic pulses with mirrored polarity and the same pulse width without an inter-pulse delay. The observation is the cross-membrane voltage over time observed at a node at a fixed distance from the stimulation electrode. For each input, the response is a vector of real values T that represent the amplitude over time at the selected node. The characteristics of these waveforms, such as the time-to-peak, amplitude, decay rate, offset, and, in some cases, the presence of multiple spikes, are affected by the pulse features. Fig. 3 shows an example of a double pulse and the corresponding AP.

Table 2
Description of models and acquisition classes with potential use cases.

	Class Name	What it does	Use case(s)
Models	MCDropout	MC-Dropout wrapper for a single Keras model that provides probabilistic predictions via stochastic dropout passes.	Very large search spaces (>15 dims) or multioutput problems.
Architectures	ProbabilisticNetwork	Extension of the GaussianNetwork by Trieste for classification problems.	Low dimensional classification problems.
Acquisitions	TemporalVariance	Ranks candidates by predictive variance and selects highest-variance points (pure exploration).	Uncertainty sampling, exploring the search space to reduce model uncertainty.
	TemporalVarianceMovement	Selects points using a weighted score of mean-gradient magnitude (movement) and variance.	Uncertainty sampling for time series, by taking the gradient it highlights high errors in segments of the series with high variance.
	MaxCoverageSampler	Selects points maximizing minimal distance to observed points (coverage).	Baseline for evaluation. Intuitive way of sampling by trying to fill the empty spaces in the search space.
	DiversityUncertaintySampler	Balances normalized uncertainty and coverage via a beta weight. Iteratively selects points combining both scores.	Extension made to improve BALD in scenarios where it oversamples in the same region.
	DiscreteBatchSampling	Samples a random pool from the search space, ranks via a sampler, and returns top-K points according to a given metric.	A flexible function where an arbitrary criterion in the inputs, outputs, or both can be selected. Only intended for problems where evaluation can be done over thousands of candidates fast.
	ExpectedImprovementXi	Acquisition function that uses a ξ parameter to trade off exploration vs exploitation.	Scenarios where the exploration/exploitation needs to be tuned during data collection.

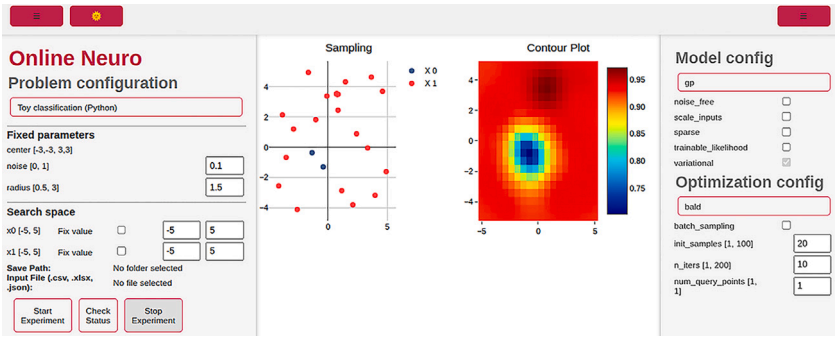

Fig. 2. GUI of OnlineNeuro. The GUI presents a left panel for problem configuration, a right panel for model and optimizer configuration, and a center panel for displaying results. The problem being solved corresponds to the circle toy problem, and shows the collected samples and the posterior mean of the model during runtime.

Table 3
Fixed parameters of the AxonSim simulation.

Component	Parameter	Value	Parameter	Value
Pulse	Number of electrodes	1	Electrode position	[−1 mm, −1 mm]
	Electrode type	Single	Frequency	Single
	Pulse type	Biphasic	–	–
Axon	Curvature	0°	Medium resistance	35 Ω cm^{-1}
	Diameter	12 µm	Length	40 mm
Model	Model type	McIntyre-Richardson-Grill (MRG)		

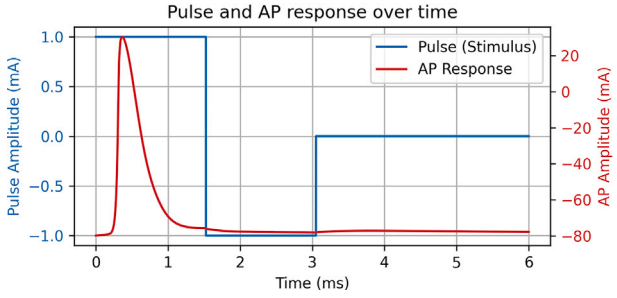

Fig. 3. Examples of the waveforms of a double pulse and the corresponding action potential.

The meta-model used is an ANN with three layers containing [32, 64, 128] units. Rectified Linear Unit (ReLU) activations are used for the hidden layers, and linear activation is used for the output layer. All layers use dropout (rate 0.3) during training for regularization and during inference for uncertainty estimation. The network parameters are optimized using the Adam optimizer with a learning rate of 0.001. The acquisition function is Temporal Variance-Movement with $\alpha = 0.1$.

The initial samples are collected using Sobol sampling. The samples are divided into an initial training dataset and a holdout dataset for validation purposes. After the initial fit, the active learning optimization is run for a total of 100 additional samples. Fig. 4 shows the root mean squared error (RMSE) of the training and validation datasets. Notice that the training set contains samples that the model has directly observed and is only used as a reference, while the holdout dataset contains points that are not included in the training. Fig. 5 shows examples of the final prediction of both validation datasets. Finally, Fig. 6 shows the initial samples, the collected samples, and the final uncertainty estimate. This figure shows how the active learning strategy focused on the edges of the search space, as these contain the APs with the highest variation in amplitude.

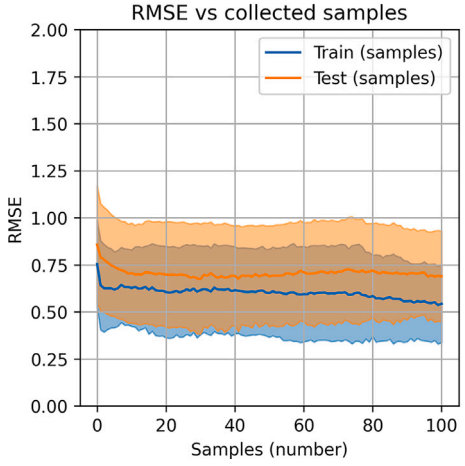

Fig. 4. Mean RMSE over the holdout samples and the initial fit samples. The intervals correspond to the error variance across samples. Convergence seems to be achieved after collecting 30 samples, with little to no improvement afterwards.

4.3. Action potential propagation blocking in cajal

This example illustrates the application of our package to optimize the parameters of a blocking pulse electrode, specifically, amplitude, pulse width, and delay to inhibit the propagation of APs in a myelinated axon model. Precise AP blocking is essential for advancing applications in functional electrical stimulation and neuromodulation. The reproducible code for this can be found in `notebooks/GP_Cajal_AP_blocking.ipynb`.

The experiment simulates the stimulation responses of a myelinated axon with 27 µm diameter and 120 mm length. The setup consists of two electrodes: a stimulation electrode injects a monophasic pulse to trigger an AP, and a blocking electrode injects a monophasic pulse to stop the AP propagation. The spacing between electrodes is 60 mm. Both electrodes are placed 1 mm above the axon (extramembrane stimulation). The fixed parameters are presented in Table 4. The parameters of the stimulation electrode are fixed and selected based on single-electrode simulations in which the configuration triggers an AP. Notice that when multiple electrodes are simulated, their fields interact, and the fixed configuration may not necessarily trigger the AP. In this evaluation, these events are considered as non-blocked events, as no initial AP was generated. The optimization parameters are the amplitude, pulse width, and delay of the

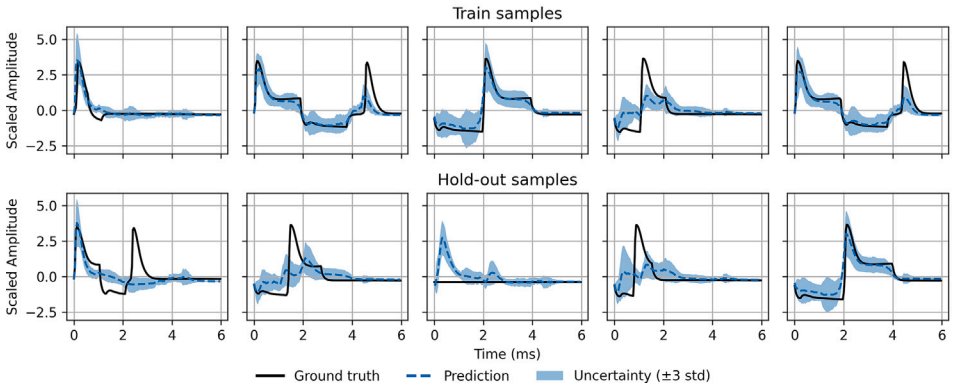

Fig. 5. Examples of AP responses of the training and out-of-bag validation datasets and predictions with uncertainty estimation after the model optimization. Black represents the actual values and dashed blue corresponds to the predictions and the uncertainty estimate (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.).

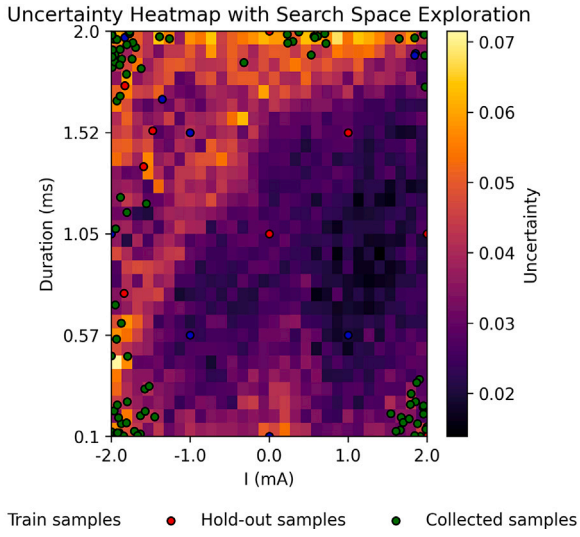

Fig. 6. Model's uncertainty after training, initial samples for training and evaluation, and collected samples. The initial training samples and holdout samples were selected using Sobol method. Subsequent points are sampled using the optimization strategy.

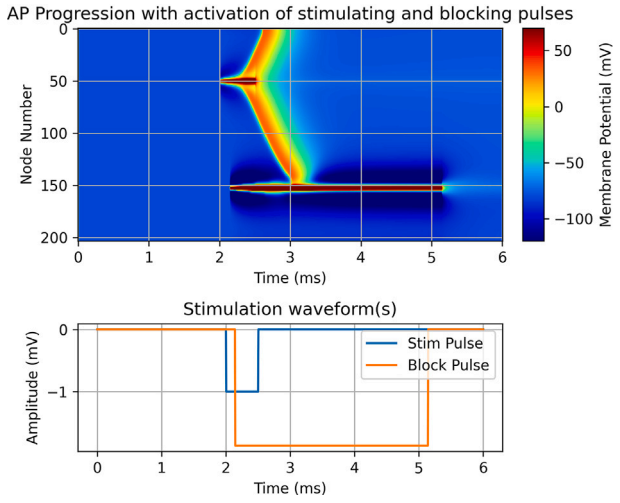

Fig. 7. Top: Example of an AP blocked at node 150. Bottom: The waveforms that generate and block the AP.

blocking electrode. The parameter search space was defined as follows: amplitude range from -4mA to 4mA, pulse width from 0.05s to 3s, and delay from 0.05s to 3s. A trial is considered valid (label = 1) if an AP is generated and effectively blocked. Fig. 7 illustrates the spatiotemporal propagation of an AP blocking around node 150. In order to enhance the parameter search, these problems can be explored with the GUI. Fig. 8 shows the results over the dynamic interface.

For the active-learning strategy, the meta-model is a variational Gaussian process with Bayesian Active Learning by Disagreement (BALD) as acquisition function [12]. This strategy is benchmarked against three baselines: fixed grid search with spacing 5 per dimension (5^d with $d = 3$, sampled without replacement), MaxiMin coverage sampling, and uniform random sampling (a more extensive discussion of these is provided in the supplementary material).

The initial samples (30) are collected using Sobol sampling. After the initial fit, the active learning optimization is run for an additional 100 samples. Fig. 9 presents the cumulative count of positive samples (blocked AP) during sampling for each method. In this figure, we can appreciate that active learning is able to find valid configurations relatively easily despite it being an imbalanced problem where the positive class is in the minority. Fig. 10 illustrates the initial and observed samples for each of the sampling methods, the colors indicate whether an AP is blocked or not. This figure shows how the active learning strategy

Table 4
Fixed parameters of the Cajal simulation.

Component	Parameter	Value	Parameter	Value
Pulse	Positions	[−30 mm, −1 mm]; [30 mm, −1 mm]	Frequency	Single pulse
	Stimulation waveform	Monophasic	Electrode type	Single
	Blocking waveform	Monophasic	–	–
Axon	Diameter	27 µm	Length	120 mm
Model	Model type	McIntyre-Richardson-Grill (MRG)		

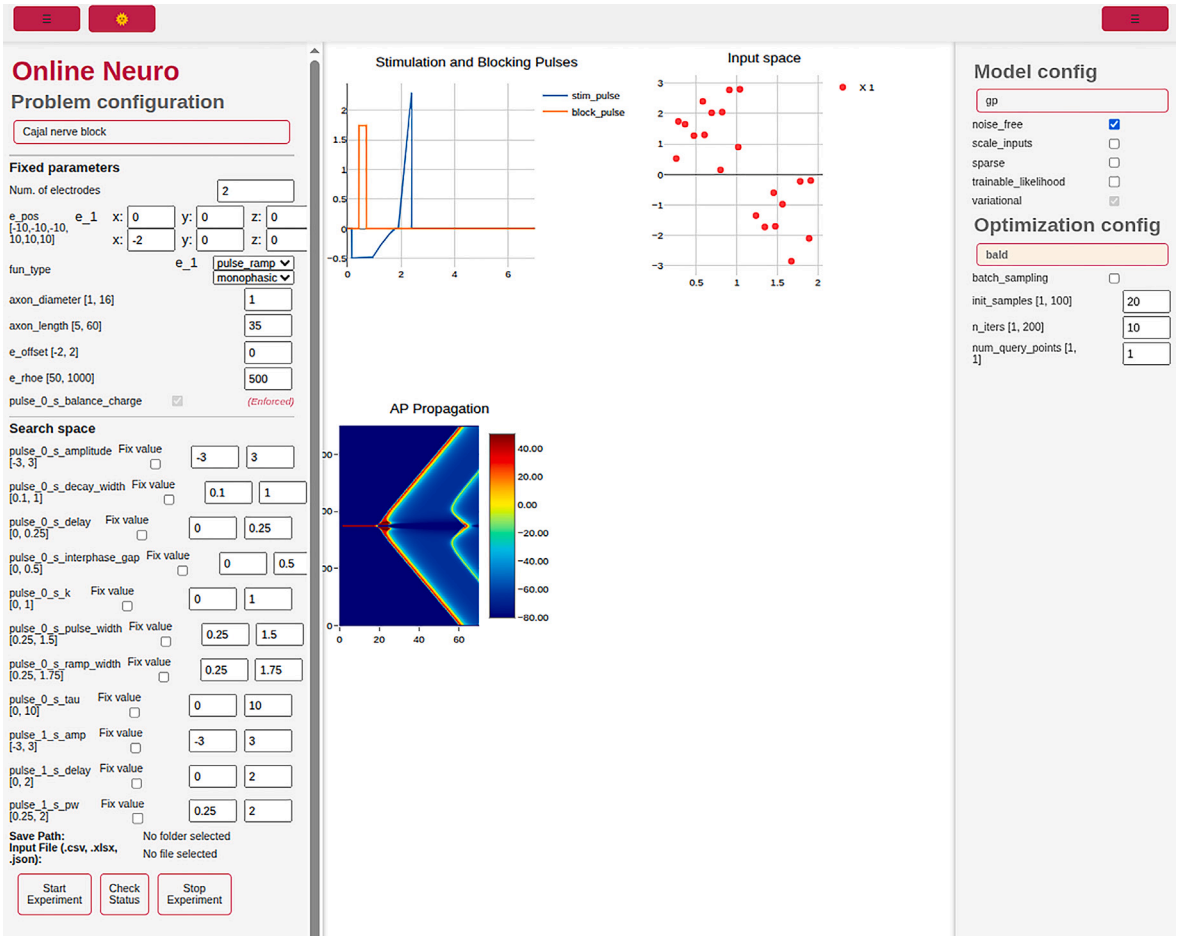

Fig. 8. Example of the AP block problem through the GUI, allowing for dynamic visualization of pulses, search space and action potential propagation over time.

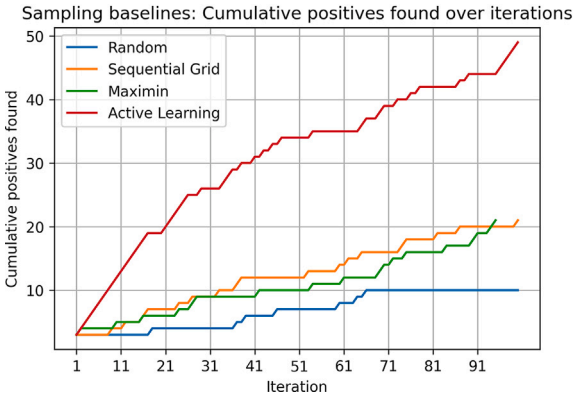

Fig. 9. Cumulative count of configurations that effectively block the AP propagation during the search space exploration for different methods.

focuses on exploring the decision boundaries, and it is more effective at finding blocking configurations in the process. Fig. 7 shows an example of a blocked AP.

Final parameter selection can follow two routes: use the best observed feasible configuration, or use the surrogate model to infer system behavior and localize predicted optima. An example of this is included in `notebooks/GP_Cajal_sinusoid_delay.ipynb`.

5. Impact

OnlineNeuro is a library designed to facilitate sequential optimization of neurostimulation parameters in an efficient and user-friendly manner by integrating computational models with active learning techniques. Active learning in neural stimulation has been investigated for clinical studies in memory function [3], motor control [7,15,16], and epilepsy treatment [18], among others, and is likely to continue developing with recent advances in implantable devices and deep learning. Despite this progress, there remains a need for open frameworks that support these initiatives under a common structure. More recently, a Bayesian optimization framework (OBOES) for evoked signals during stimulation has been presented [22]. Its dynamic dashboard allows users to explore simulated vagus nerve stimulation data using BO techniques; however, it currently does not support analysis of external datasets. OnlineNeuro is positioned as a bridge between active-learning tools and neural simulation platforms. Its long-term vision includes the inclusion of novel active-learning methods and extension of utility functions for neurosimulation packages such as AxonSim [6], Cajal, and AxonML [13], with the ultimate goal of enabling efficient in-vivo studies.

The potential benefits of OnlineNeuro include faster evaluation of electrode configurations, discovery of novel stimulation protocols, and more efficient data collection.

As future work, this library offers opportunities for research in active learning and closed-loop neurostimulation. We highlight:

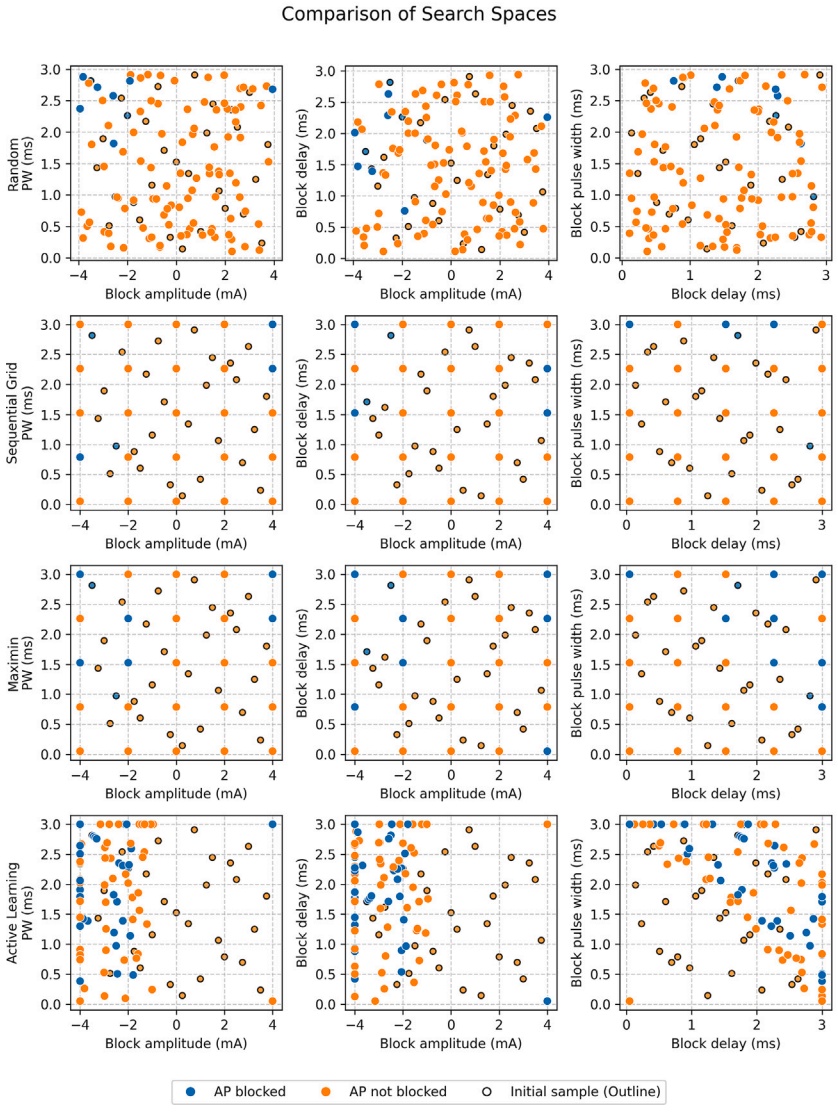

Fig. 10. Initial and collected samples and observed outputs for the AP block experiments for different search strategies.

- Selective AP blocking in axon populations, where the goal is to generate APs and block their propagation only on specific axon bundles. This scenario addresses diversity in neuron fiber physical properties (e.g., width), their sensitivity to pulse configurations, and the spatial distribution of fibers.
- Modeling of complex stimulation patterns and dosing over longer periods, including continuous and burst stimulation, as well as effects across multiple sessions.
- Safe evaluation of novel stimulation techniques with limited side-effect risk by extending OnlineNeuro to multi-objective and constrained optimization.
- Adaptation to long-term plasticity through models that can be updated as neural dynamics change.
- Short-term in-vivo validation studies with acute responses to assess active-learning protocols. For example, cardio-respiratory regulation through vagus nerve stimulation (VNS), as investigated in OBOES studies [22], provides physiological feedback within seconds.
- Long-term in-vivo studies targeting physiological modulation over extended periods, including interventions whose effects emerge over days or weeks, such as transcranial magnetic stimulation (TMS) in epilepsy treatment.

6. Conclusions

We present OnlineNeuro, a Python based framework for active learning in the context of neurostimulation experiment optimizations. The library is compatible with three popular neural simulators: AxonSim, Cajal, and AxonML; and it contains routines and models to optimize regression and classification problems, bringing the benefits of active learning to neurostimulation optimization problems.

OnlineNeuro can be used for multiple research purposes, such as the creation of meta-models to better understand axon dynamics, investigating the spatial and temporal effects of different electrode configurations and stimulation parameters. The active learning framework allows for more effective data collection.

Finally, its dedicated GUI aims to make interaction with the library and live analysis seamless. In addition, the interface handles common tasks such as problem configuration, model saving, results reporting, and interactive visualizations.

CRediT authorship contribution statement

Diego Nieves Avendano: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Project administration, Methodology, Investigation, Formal analysis, Conceptualization. **Arne**

Callaert: Writing – review & editing, Validation, Methodology, Formal analysis. **Philipp Schnepel:** Writing – review & editing, Validation, Project administration, Conceptualization. **Vojkan Mihajlovic:** Writing – review & editing, Supervision, Project administration, Funding acquisition, Conceptualization. **Femke Ongenae:** Writing – review & editing, Project administration, Funding acquisition, Conceptualization. **Sofie Van Hoecke:** Writing – review & editing, Supervision, Project administration, Funding acquisition, Conceptualization.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During the preparation of this work, the authors used Google AI Studio in order to review and improve the writing style of the manuscript, and help generating the documentation for the repository. After using this service, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Part of this research is funded through the imec AAA project on Autonomous Therapeutics.

Appendix A. Supplementary data

Supplementary data to this article can be found online at doi:10.1016/j.softx.2026.102648.

Extended annex content (toy-benchmark definitions, additional figures, and the full Temporal Variance–Movement formulation) is provided in repository supplementary notebooks and is not part of the main manuscript word count.

The supplementary toy benchmarks include Gaussian-process and random-forest baselines implemented with standard scikit-learn tooling [19].

References

[1] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X. TensorFlow: large-scale machine learning on heterogeneous systems; 2015. https://www.tensorflow.org/. Software available from tensorflow.org.

[2] Aiello G, Valle G, Raspopovic S. Recalibration of neuromodulation parameters in neural implants with adaptive Bayesian optimization. J Neural Eng Apr 2023;20(2):026037. ISSN 1741-2552, https://doi.org/10.1088/1741-2552/acc975.

[3] Ashmaig O, Connolly M, Gross RE, Mahmoudi B. Bayesian optimization of asynchronous distributed microelectrode theta stimulation and spatial memory. In: 2018 40th annual international conference of the IEEE engineering in medicine and Biology society (EMBC); 2018. p. 2683–6. https://doi.org/10.1109/EMBC.2018.8512801

[4] Bellman RE. Adaptive control processes: a guided tour. Princeton University Press. Dec 1961. https://doi.org/10.1515/9781400874668. ISBN 9781400874668.

[5] Caulfield KA, Brown JC. The problem and potential of tms' infinite parameter space: a targeted review and road map forward. Front Psychiatry May 2022;13. ISSN 1664-0640, https://doi.org/10.3389/fpsyt.2022.867091.

[6] Danner SM, Wenger C, Rattay F. Electrical stimulation of myelinated axons: an interactive tutorial supported by computer simulation. VDM Verlag Dr. Müller. Aug 2011. ISBN 9783639370829.

[7] Desautels TA, Choe J, Gad P, Nandra MS, Roy RR, Zhong H, Tai Y-C, Edgerton VR, Burdick JW. An active learning algorithm for control of epidural electrostimulation. IEEE Trans Biomed Eng 2015;62(10):2443–55. https://doi.org/10.1109/TBME.2015.2431911

[8] Famm K, Litt B, Tracey KJ, Boyden ES, Slaoui M. A jump-start for electroceuticals. Nature Apr 2013;496:159–61. https://doi.org/10.1038/496159a

[9] Frazier PI. A tutorial on Bayesian optimization; 2018. https://arxiv.org/abs/1807.02811.

[10] Gal Y, Ghahramani Z. Dropout as a Bayesian approximation: representing model uncertainty in deep learning. In: Balcan MF, Weinberger KQ, editors. Proceedings of the 33rd international conference on machine learning, volume 48 of proceedings of machine learning research. New York, New York, USA: PMLR; 20–22 Jun 2016. p. 1050–9. https://proceedings.mlr.press/v48/gal16.html.

[11] Hines ML, Carnevale NT. The neuron simulation environment. Neural Comput Aug 1997;9(6):1179–209. ISSN 1530-888X, https://doi.org/10.1162/neco.1997.9.6.1179.

[12] Houlsby N, Huszár F, Ghahramani Z, Lengyel M. Bayesian active learning for classification and preference learning; 2011. https://arxiv.org/abs/1112.5745.

[13] Hussain MA, Grill WM, Pelot NA. Highly efficient modeling and optimization of neural fiber responses to electrical stimulation. Nat Commun Aug 2024;15(1):7597. ISSN 2041-1723, https://doi.org/10.1038/s41467-024-51709-8.

[14] Hüllermeier E, Waegeman W. Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods - machine learning; Mar 2021. https://link.springer.com/article/10.1007/s10994-021-05946-3.

[15] Laferrière S, Bonizzato M, Côté SL, Dancause N, Lajoie G. Hierarchical Bayesian optimization of spatiotemporal neurostimulations for targeted motor outputs. IEEE Trans Neural Syst Rehabil Eng 2020;28(6):1452–60. https://doi.org/10.1109/TNSRE.2020.2987001

[16] Losanno E, Badi M, Wurth S, Borgognon S, Courtine G, Capogrosso M, Rouiller EM, Micera S. Bayesian optimization of peripheral intraneural stimulation protocols to evoke distal limb movements. J Neural Eng Dec 2021;18(6):066046. ISSN 1741-2552, https://doi.org/10.1088/1741-2552/ac3f6c.

[17] Matthews AGDG, van der Wilk M, Nickson T, Fujii K, Boukouvalas A, León-Villagrá P, Ghahramani Z, Hensman J. GPflow: a Gaussian process library using TensorFlow. J Mach Learn Res Apr 2017;18(40):1–6. http://jmlr.org/papers/v18/16-537.html.

[18] Park S-E, Connolly MJ, Exarchos I, Fernandez A, Ghetiya M, Gutekunst C-A, Gross RE. Optimizing neuromodulation based on surrogate neural states for seizure suppression in a rat temporal lobe epilepsy model. J Neural Eng Jul 2020;17(4):046009. ISSN 1741-2552, https://doi.org/10.1088/1741-2552/ab9909.

[19] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E. Scikit-learn: machine learning in Python. J Mach Learn Res 2011;12:2825–30.

[20] Picheny V, Berkeley J, Moss HB, Stojic H, Granta U, Ober SW, Artemev A, Ghani K, Goodall A, Paleyes A, Vakili S, Pascual-Diaz S, Markou S, Qing J, Loka NRBS, Couckuyt I. Trieste: efficiently exploring the depths of black-box functions with tensorflow; 2023. https://arxiv.org/abs/2302.08436.

[21] Plotly Technologies Inc. Collaborative data science; 2015. https://plot.ly.

[22] Wernisch L, Edwards T, Berthon A, Tessier-Lariviere O, Sarkans E, Stoukidi M, Fortier-Poisson P, Pinkney M, Thornton M, Hanley C, Lee S, Jennings J, Appleton B, Garsed P, Patterson B, Buttinger W, Gonshaw S, Jakopec M, Shunmugam S, Mamen J, Tukiainen A, Lajoie G, Armitage O, Hewage E. Online Bayesian optimization of vagus nerve stimulation. J Neural Eng Apr 2024;21(2):026019. ISSN 1741-2552, https://doi.org/10.1088/1741-2552/ad33ae; Repository DOI: https://doi.org/https://10.0.20.161/zenodo.18434888 .