Asynchronous Multi-fidelity Hyperparameter Optimization Of Spiking Neural Networks

Thomas Firmin, Pierre Boulet, El-Ghazali Talbi

Asynchronous Multi-fidelity Hyperparameter Optimization Of Spiking Neural Networks

Preprint accepted at the International Conference On Neuromorphic Systems 2024 (ICONS 2024)
https://iconsneuromorphic.cc/

Thomas Firmin*, Pierre Boulet†, El-Ghazali Talbi‡
CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, Univ. Lille
Campus scientifique, Bâtiment ESPRIT, Avenue Henri Poincaré
Lille, France F-59000
Email: *thomas.firmin@univ-lille.fr, †pierre.boulet@univ-lille.fr, ‡el-ghazali.talbi@univ-lille.fr

***Abstract*—Spiking Neural Network (SNN) are peculiar networks based on the dynamics of timed spikes between fully asynchronous neurons. Their design is complex and differs from usual artificial neural networks as they are highly sensitive to their hyperparameters. Some SNNs are unable to emit enough spikes at their outputs, causing a more challenging, even an impossible, learning task. Such networks are called *silent networks*. By considering mistuned hyperparameters and architecture, this concept describes a generalization of the signal-loss problem. In this work, to accelerate the hyperparameter optimization of SNNs trained by surrogate gradient, we propose to leverage silent networks and multi-fidelity. We designed an asynchronous black-box constrained and cost-aware Bayesian optimization algorithm to handle high-dimensional search spaces containing many silent networks, considered as infeasible solutions. Large-scale experimentation was computed on a multi-nodes and multi-GPUs environment. By considering the cost of evaluations, we were able to quickly obtain acceptable results for SNNs trained on a small proportion of the training dataset. We can rapidly stabilize the inherent high sensitivity of the SNNs' hyperparameters before computing expensive and more precise evaluations. We have extended our methodology for search spaces containing 21 and up to 46 layer-wise hyperparameters. Despite an increased difficulty due to the higher dimensional space, our results are competitive, even better, compared to their baseline. Finally, while up to 70% of sampled solutions were silent networks, their impact on the budget was less than 4%. The effect of silent networks on the available resources becomes almost negligible, allowing to define higher dimensional, more general and flexible search spaces.**

I. Introduction

Spiking Neural Networks (SNNs) are analog neural networks closer to the biology compared to their relative digital Artificial Neural Networks (ANNs). SNNs use time for computations via sparse events named spikes [1]. The peculiar characteristics of SNNs make them suitable for low-power applications and easily scalable thanks to the inherent neurons' asynchronicity [2], [3]. However, they cannot be directly computed on usual Von Neumann architectures. Instead, SNNs are run on specific neuromorphic hardware, such as SpiNNaker [4]. One can also compute SNN on regular CPU or GPU hardware by using simulators, such as Lava[1]. These specificities contribute to a rich choice of hyperparameters (HPs), notably concerning the neuron's model (like the ANN activation function) [5], [6]. However, a counterpart is that SNNs are known to be very sensitive to their numerous HPs [5], [7]–[9], making their HyperParameter Optimization (HPO) challenging.

HPO of machine learning models is a well-defined problem [10], yet tuning HPs of SNNs with the same methodology as with ANN, appears to be inefficient when high dimensional search-spaces are considered [5]. We define the HPO via the maximization of the classification accuracy on a test set of a trained SNN $\mathcal{N}$, *hyperparameterized* by λ, as:

$$\lambda^\star \in \underset{\lambda\in\Lambda}{\operatorname{argmax}}\, Accuracy(\mathcal{N}^\lambda_{\theta^\star}, \mathcal{D}_{\text{valid}}), \text{ s.t. } c_1 \leq 0, c_2 \leq 0, ... \tag{1}$$

where $\theta^\star$ are the optimal parameters (e.g. weights) obtained by training $\mathcal{N}$ on a training set $\mathcal{D}_{\text{train}}$. The bounded search space, Λ contains all HPs combinations λ. The theoretical optimum, $\lambda^\star$, is defined according to the intermediate performances computed on the validation dataset, $\mathcal{D}_{\text{valid}}$. As in [5], [11], we consider black-box constraints defined by $c_i \leq 0$. These constraints model a minimum required per-sample spiking activity for a certain proportion of the dataset within a given layer i. In this work, we only consider a constraint c_{out} on the output layer of $\mathcal{N}$. For good practice [12], the final performances and generalization capability of $\mathcal{N}^{\lambda^\star}_{\theta^\star}$ are assessed using a hold-out test dataset written $\mathcal{D}_{\text{test}}$.

The computational complexity of training a SNN can be explained by several HPs [9] (e.g., train set size, epochs, batch size, number of neurons, etc.), but also by specific fidelity HPs, such as the duration or the number of encoding frames of a single sample from the dataset. Thus, we also consider a cost function $Cost : \mathcal{N}^\lambda \rightarrow \mathbb{R}^+$ defining the positive computational cost (e.g., training time) of a SNN. In this work, a multi-fidelity Bayesian Optimization (BO) algorithm leverages $Cost$ to accelerate the search of $\lambda^\star$.

The main contributions of the paper are as follows:

- The design of a high dimensional search space, from dimension 21 to 46, including fidelity hyperparameters, and containing many silent networks.
- A significant decrease in the impact on the time budget of silent networks while maintaining competitive performances.

[1]https://github.com/lava-nc/lava-dl

- A Cost-Aware Scalable Constrained Bayesian optimization (CASCBO) algorithm using Thompson sampling on the greedy improvement per unit with an asynchronous trust region.
- An improved constraint formulation linked to the early stopping criterion to reduce the computational cost of selecting silent networks within a high dimensional search space.
- A parallel asynchronous implementation of our approach on clusters of GPUs (Grid'5000 [13]).

The paper is organized as follows. In Section II works tackling HPO of SNNs and multi-fidelity bayesian optimization are presented. Subsequently, in Section III, we present the applied methodology, including a modified version of the early stopping and our Cost Aware Scalable Constrained Bayesian Optimization algorithm to handle the costs of evaluating SNNs. In Sections IV and V, we discuss experimental results and the effects of the fidelity HPs on the optimization process. We give the main conclusions of this work and some perspectives in Section VI.

II. Background and Related works

A. Multi-fidelity and spiking neural networks

In [5], the Scalable Constrained Bayesian Optimization algorithm (SCBO) [11] was applied to HPO of 4 SNNs trained by Spike Timing Dependent Plasticity (STDP) [14] and SLAYER [15], to classify Poisson Encoded MNIST [16] and DVS-Gesture [17]. The authors applied an indirect spike-based early stopping criterion to early detect *silent networks* and avoid costly and worthless computations. A silent network is a SNN unable to output enough spikes for a given task because of mistuned HPs or architecture. This concept is a generalization of the signal loss problem [18], [19] explained by a too deep SNN. Because spiking datasets have a heterogeneous spiking activity, for and within each class, the early stopping is based on a per-sample spiking activity. If one can detect that, a proportion β_{train} of samples within the training dataset $\mathcal{D}_{\text{train}}$ outputting less than α spikes, is superior to a given proportion β ($\beta_{\text{train}} \geq \beta$), then the training is stopped and the SNN is considered as a silent network. So, a constraint as described in 1 can be written, $c_{\text{out}} \leq 0 \iff \beta_{\text{train}} - \beta \leq 0$. A negative or positive value of c_{out}, is an indication that the training phase has been stopped or fully completed. Therefore, the constraint forces SCBO to find HPs combinations carrying out the training process. Furthermore, results in [5], emphasize that some very early stopped SNNs can have acceptable performances ($\approx 80\%$ accuracy), leading the way to even more accelerated HPO by considering multi-fidelity, i.e., training on smaller subsets of $\mathcal{D}_{\text{train}}$.

Hyperparameters definitely have an impact on the computation time of SNNs. In [9], $Cost$ was considered within a multi-objective approach so to maximize the accuracy while minimizing the training time per epoch. The authors have demonstrated that high accuracies can be obtained in lower training time. As a result, intensive and expensive training is not necessary to obtain competitive accuracies.

Another work [20] used Bayesian Optimization HyberBand (BOHB), a multi-fidelity Bayesian Optmization algorithm based on the HyperBand algorithm [21]. Authors optimized a 3 HPs search space (leakage, time-steps, learning rate) of the S-Resnet38 architecture applied to CIFAR-10 and CIFAR-100. However, the impact of the multi-fidelity on the optimization process (the training set size) was not investigated. BOHB maximizes the fidelity HP, i.e. the size of the training subset from $\mathcal{D}_{\text{train}}$, by doing the assumption that higher fidelity always results in better performances [10]. However, as described in [10], multi-fidelity HPO can be divided into two groups, one for which the previous assumption holds, and another one where higher fidelity does not necessarily result in better performances, notably in the case of overfitting.

In our SNNs problem, specifically when they are simulated [22], [23], several HPs can control a certain fidelity, i.e., complexity of the simulator. In BindsNet [22], the temporal granularity, named dt, in milliseconds, controls the number of time steps during the simulation. In Brian2 [23], one can choose between different numerical integration methods such as Euler or Runge-Kutta algorithms, which also affects the performances [7]. In general, and for non-event-driven simulators, spikes of a sample are accumulated within a certain number of frames. This HP also has implications on the performances of the SNN. The higher the number of frames, the higher the computation time, but not necessarily the better the performances [24], [25].

B. Asynchronous and cost aware BO

Because of the early stopping criterion described in [5], asynchronous parallelization is unavoidable, as silent networks can be detected in a few seconds (on Poisson encoded MNIST), while fully trained SNNs can last a few hours. To tackle asynchronous BO several techniques exist in the literature. For instance, fantasizing [26] consists in using the Gaussian Process posterior distribution to impute values of pending evaluations, and including them in the optimization of the acquisition function without retraining the Gaussian process for every fantasy. Another and simpler solution is to use asynchronous Thompson Sampling [27] which replaces the acquisition function by the maximization of a batch of samples from the Gaussian process posterior distribution. Such an approach is cheaper compared to fantasies and local penalization [28], as it does not involve the optimization of an acquisition function. Therefore, Thompson Sampling appears to be a reliable strategy, particularly when our search space may contain many silent networks [5]. These silent networks can be quickly detected and evaluated, involving numerous queries to obtain new batches of solutions in a short period of time.

Additionally, Thompson sampling is used in Trust Region Bayesian Optimization (TuRBO) [29] and in its constrained version known as SCBO [11]. SCBO allows tackling HPO for high dimensional and black-box constrained problems. Such an approach was successfully applied in [5] for optimizing hyperparameters of SNNs while leveraging silent networks.

Nonetheless, difficulties arise when dealing with multi-fidelity. A solution to this are *Multi-output Gaussian processes* (MOGP), allowing to jointly model, with a single Gaussian process, all fidelities [28]. MOGP allows the transfer of information between levels of fidelities. In [28], the authors proposed a multi-fidelity approach to TuRBO implying discrete fidelities, modeled by a MOGP outputting a M-dimensional mean and a $M \times M$ co-variance matrix for each sample from the posterior, where M is the number of fidelities. A major drawback of this approach is the discretization of a single HP describing the fidelity. We saw that when SNNs are simulated the fidelity can be expressed as a combination of different HPs which can be continuous, and might sometimes result in overfitting. Thus, a solution to our problem is to consider a generalization of multi-fidelity Bayesian optimization, known as Cost-Apportionned Bayesian Optimization (CArBO) [30]. Here, the idea is to train two Gaussian processes, one for estimating the actual objective function, and a second to impute the computational cost of a HPs combination. The algorithm then optimizes an acquisition function known as the cost-cooling Expected Improvement per unit (EIpu):

$$\text{EI-cool}(\lambda) := \frac{\text{EI}(\lambda)}{\tilde{C}(\lambda)^{\kappa}} \ , \tag{2}$$

where λ is a HPs combination and $\tilde{C}$ the positive mean from the Gaussian process posterior distribution (on the costs) for a given sample λ. The annealing parameter κ, decreasing from 1 to 0, describes the contribution of the cost on the EIpu, allowing to focus on cheap computation at the beginning and on expensive ones by the end of the remaining budget. When $\kappa \to 0$, the acquisition function becomes the usual Expected Improvement (EI). Unfortunately, a major drawback of EIpu is its difficult tractability toward asynchronous parallelization.

III. Methodology

A. Improved early stopping and constraints

The early stopping criterion, described in [5], is based on two HPs, α describing the minimum number of output spikes for a single sample from $\mathcal{D}_{\text{train}}$, and β the maximum acceptable proportion of samples outputting less than α spikes. During the training, after each batch, the proportion β_{train} of non-spiking samples is computed. If $\beta_{\text{train}} \geq \beta$, the training is stopped. We can extend this to $\lambda^{\text{train}} \in]0, 1]$, an HP defining the proportion of the initial $\mathcal{D}_{\text{train}}$ used for training. To even more accelerate the detection of silent networks, β now describes the proportion of samples from the subset $\mathcal{D}'_{\text{train}}$ of $\mathcal{D}_{\text{train}}$, outputting less than α spikes. So, as described in alg. 1, which can be easily extended to batches of data, the early stopping is now based on the subset of $\mathcal{D}_{\text{train}}$ proportional to λ^{train}. Thus, selecting a convenient β relies on the minimum value of λ^{train} and on the size of $\mathcal{D}_{\text{train}}$. Moreover, instead of computing a constraint on the proportion of samples outputting **less than** α spikes, $c_{\text{out}} := \beta_{\text{train}} - \beta$, we rewrite it as the proportion β'_{train} of samples that have outputted **at least** α spikes before the network being stopped:

$$c_{\text{out}} := 1 - \beta'_{\text{train}} - \beta \ . \tag{3}$$

Algorithm 1 SNN training with improved constraint

Inputs:
- $\mathcal{N}_\theta^\lambda$ — *Network*
- $\mathcal{D}_{\text{train}}$ — *Training data*
- λ^{train} — *Proportion of $\mathcal{D}_{train}$*
- epochs — *Number of epochs*
- α — *Minimum spiking activity*
- β — *Maximum proportion of non spiking samples*

Outputs: $\mathcal{N}_{\theta^\star}^\lambda,\ \beta'_{\text{train}}$

$\mathcal{D}'_{\text{train}} \leftarrow \texttt{SUBSET}(\mathcal{D}_{\text{train}}, \lambda^{\text{train}})$ *Exctract $\lambda^{train}\%$*
$out \leftarrow \emptyset$ *Output spikes*
$i, e \leftarrow 1$
while $(\beta_{\text{train}} \leq \beta) \wedge (e \leq \text{epochs})$ **do**
  $nscount \leftarrow 0$ *Number of non spiking samples*
  $scount \leftarrow 0$ *Number of spiking samples*
  while $(\beta_{\text{train}} \leq \beta) \wedge (i \leq |\mathcal{D}'_{\text{train}}|)$ **do**
    $out \leftarrow \text{Train}(\mathcal{N}_\theta^\lambda, \mathcal{D}'_{\text{train}}[i])$
    if $\texttt{SUM}(out) < \alpha$ **then** *Number of output spikes*
      $nscount \leftarrow nscount + 1$
      $\beta_{\text{train}} \leftarrow \frac{nscount}{|\mathcal{D}'_{\text{train}}|}$ *Ratio of non spiking samples*
    else
      $scount \leftarrow scount + 1$
      $\beta'_{\text{train}} = \frac{scount}{|\mathcal{D}'_{\text{train}}|}$ *Ratio of spiking samples*
    $i \leftarrow i + 1$
  $e \leftarrow e + 1$
return $\mathcal{N}_{\theta^\star}^\lambda,\ \beta'_{\text{train}}$

So we obtain a stochastic value (depending on the shuffling of the dataset), describing how far the training of $\mathcal{N}$ went before being stopped. If the training of $\mathcal{N}$ was not stopped, then $c_{\text{out}} < 0$. A hypergeometric distribution can model the probability of encountering non-spiking samples within the first n samples from D'_{train} during training. If X is the number of non-spiking samples encountered after n draws, then $X \sim \text{Hypergeometric}(n, K, |\mathcal{D}'_{\text{train}}|)$. However, we cannot know a-priori the values of K, i.e., the number of non-spiking samples in D'_{train}. Moreover, this value K may vary through epochs. Thus, in probability, the higher K, the lower β'_{train} and the sooner $\mathcal{N}$ should be stopped. Therefore, one of the objectives of the optimization algorithm is to minimize c_{out} to ensure a minimal per-sample spiking activity.

B. Cost aware Thompson sampling

In the following section we describe our modification of the Scalable Constrained Bayesian Optimization (SCBO) [11], specifically designed for our HPO problem, so to handle the cost of evaluations. We named this algorithm Cost Aware Scalable Constrained Bayesian Optimization (CASCBO).

In SCBO [11], Thompson sampling replaces the acquisition function and is adapted to black-box constraints. In this work, the accuracy, the constraint on the outputs and

the cost are modelled by their respective Gaussian process. For simplicity, we write the performances of a network as $f(\lambda) := Accuracy(\mathcal{N}^{\lambda}_{\theta^*}, \mathcal{D}_{\text{valid}})$, the value of the constraint for the HPs combination λ is written $c_{\text{out}}(\lambda)$, and $Cost(\lambda)$ is the positive computational cost of training $\mathcal{N}^{\lambda}$. Therefore, CASCBO maintains 3 Gaussian Processes for the 3 previous functions. The resulting prior distributions are written :

$$\begin{aligned} f(\lambda^{1:k}) &\sim \mathcal{GP}\left(\mu_0^f(\lambda^{1:k}), \Sigma_0^f(\lambda^{1:k})\right) \ , \\ c_{\text{out}}(\lambda^{1:k}) &\sim \mathcal{GP}\left(\mu_0^{c_{\text{out}}}(\lambda^{1:k}), \Sigma_0^{c_{\text{out}}}(\lambda^{1:k})\right) \ , \\ Cost(\lambda^{1:k}) &\sim \mathcal{GP}\left(\mu_0^{Cost}(\lambda^{1:k}), \Sigma_0^{Cost}(\lambda^{1:k})\right) \ , \end{aligned} \tag{4}$$

where $\lambda^{1:k}$ is the archive, i.e., the sequence of input solutions $\lambda^{(1)}, \ldots, \lambda^{(k)}$. The mean functions are denoted μ_0, and Σ_0 are the kernels (here a Matèrn 5/2) for f, c_{out} and $Cost$. We write the estimators of $f(\lambda^{1:k})$, $c_{\text{out}}(\lambda^{1:k})$ and $Cost(\lambda^{1:k})$ obtained from the Gaussian processes as $\hat{f}$, $\hat{c}_{\text{out}}$ and $\widehat{Cost}$.

At each iteration, SCBO samples $\lambda_0, \ldots, \lambda_S$ candidate solutions, within a trust region $L \subseteq \Lambda$. Here we consider Λ as the unit hypercube. Thompson sampling uses the posterior distributions, $\hat{f}$, $\hat{c}_{\text{out}}$ and $\widehat{Cost}$, to sample $r = 1, \ldots, R$, realizations for all $s = 1, \ldots, S$ candidates. In the original version of SCBO, for each realization, the algorithm computes the set of feasible candidates $\widehat{F}_r := \{\lambda_s \,|\, \hat{c}_{\text{out}}(\lambda_s) < 0\}$. In this work, $\widehat{F}_r$ contains the potential non-silent HPs combinations. While $\widehat{F}_r^{\complement}$ contains potential silent networks. Then SCBO selects a potentially optimal candidate $\tilde{\lambda}_r$ for a realization r such that:

$$\tilde{\lambda}_r \in \underset{\lambda \in \widehat{F}_r \bigcup \widehat{F}_r^{\complement}}{\operatorname{argmin}} \begin{cases} -\hat{f}(\lambda) & \text{, if } \lambda \in \widehat{F}_r \\ \hat{c}_{\text{out}}(\lambda) & \text{, if } \widehat{F}_r = \emptyset \end{cases} \tag{5}$$

So, at each iteration, SCBO returns a batch of R potentially optimal solutions, selected according to 5. A high number of candidates S and the stochasticity of the estimators, $\hat{f}$, $\hat{c}_{\text{out}}$ and $\widehat{Cost}$, ensure the diversity of the selected potentially optimal candidates, $\tilde{\lambda}_{0,\ldots,R}$.

Inspired by the EIpu [30], described by 2, and to include $\widehat{Cost}$ in 5, we propose to compute a greedy improvement based on the current best solution λ^{best}, $\Delta(\lambda) = f(\lambda^{\text{best}}) - \hat{f}(\lambda)$. Then, according to the sign of $\Delta(\lambda)$, the improvement will be weighted or penalized by $\widehat{Cost}(\lambda)$. If c_{out} is positive, then it should be penalized by $\widehat{Cost}(\lambda)$.

We propose to rewrite 5 as:

If $\hat{F}_r \neq \emptyset$,

$$\tilde{\lambda}_r \in \underset{\lambda \in \widehat{F}_r}{\operatorname{argmin}} \begin{cases} \Delta(\lambda) \div \widehat{Cost}(\lambda)^{\kappa} & \text{, if } \Delta(\lambda) < 0 \\ \Delta(\lambda) \times \widehat{Cost}(\lambda)^{\kappa} & \text{, else} \end{cases} , \tag{6a}$$

otherwise ,

$$\tilde{\lambda}_r \in \underset{\lambda \in \widehat{F}_r^{\complement}}{\operatorname{argmin}} \left\{ \hat{c}_{\text{out}}(\lambda) \times \widehat{Cost}(\lambda)^{\kappa} \right. . \tag{6b}$$

So, for each realization, 6 should return the λ with the best greedy improvement per unit if $\Delta(\lambda)$ is negative. Otherwise, it should return the less costly λ compared to the potential loss of performances. If none of the realizations are feasible, then it will return the solutions with the minimum constraint violation penalized by the cost.

C. Asynchronous trust regions

TuRBO [29] and SCBO [11] implement *trust regions*. A trust region, L, is a subspace of Λ (unit hypercube), centered on the current best solution λ^{best}. In its simplest shape, L is described by a hypercube of length l. At each iteration, both algorithms return, via the posterior distributions and Thompson sampling, a batch of solutions within L. When computed with the actual objective function, this batch of evaluated solutions, written $\mathcal{B}$, is used to:

- shrink L, by $l \leftarrow l \div 2$, if after n_{fail} successive iterations the algorithm failed to improve λ^{best}
- expand L, by $l \leftarrow l \times 2$, if at each and during n_{success} successive iterations the algorithm improved λ^{best}.

At the end of each iteration, the center of L, i.e. λ^{best}, is updated if the best solution from $\mathcal{B}$ is better than λ^{best}. In this work, a solution λ_1 is better than λ_2 if:

$$\begin{cases} f(\lambda_1) > f(\lambda_2) & \text{, if } (\nu(\lambda_1) < 0) \wedge (\nu(\lambda_2) < 0) \\ c_{\text{out}}(\lambda_1) < c_{\text{out}}(\lambda_2) & \text{, if } c_{\text{out}}(\lambda_2) \geq 0 \\ Cost(\lambda_1) < Cost(\lambda_2) & \text{, else if } (f(\lambda_1) = f(\lambda_2)) \end{cases} \tag{7}$$

Concerning the parallelization, we use a flexible asynchronous approach. It consists of constantly keeping a FIFO queue of non-evaluated solutions waiting to be sent to idle worker processes. When the number of solutions within this queue is under a certain threshold, the evaluated solutions are returned, and used to update the surrogates of CASCBO. Thus, we cannot ensure that at each iteration, the trust region will be updated using a fixed batch size. Indeed, because of the early stopping and because HPs have an impact on the evaluation time, the trust region can shrink while expensive evaluations of previous batches are not yet evaluated. Thus, at a certain time t, some evaluated solutions can be located outside the current trust region.

For simplicity, we consider a single trust region, even if in TuRBO and SCBO multiple trust regions can be maintained at the same time. Because of the asynchronicity, the center of the trust region can be relocated to a solution that was sampled a few batches before, and so this solution can be outside the current L.

We write L_I the trust region at the iteration I, and L_i the trust region at a given previous iteration $i < I$. Moreover, λ_i are now solutions (HPs combinations) sampled by CASCBO at iteration i. The batch at I is written $\mathcal{B}_I$. This batch is now built with pairs (λ_i, L_i), made of evaluated solutions λ_i and their respective L_i, sampled at previous iterations i. For readability, we do not write the quintuple $(\lambda_i, L_i, f(\lambda_i), c_{\text{out}}(\lambda_i), Cost(\lambda_i))$, as $f(\lambda_i)$, $c_{\text{out}}(\lambda_i)$ and $Cost(\lambda_i)$ are known when λ_i is evaluated. So, at iteration I, $\mathcal{B}_I := \{(\lambda_{i,j}, L_i) \,|\, i < I,\ 0 \leq j \leq r\} \setminus \mathcal{B}_{0,\ldots,I-1}$.

Now that we can temporally link solutions to the previous trust regions it were sampled in, we can redefine the update

of the current L_I. When the size of the FIFO queue of non-evaluated solutions is lower than a threshold, a batch $\mathcal{B}_I$ of evaluated solutions is returned to CASCBO to compute the next iteration I. By using 7, we can determine the best $(\lambda_{i,j}, L_i) \in \mathcal{B}_\mathcal{I}$. The current trust region L_I is then updated as follows:

- If the best $\lambda_{i,j} \notin L_I$, and if $\lambda_{i,j}$ is better than λ^{best}, according to 7, the algorithm goes back-in-time such that $L_I \leftarrow L_i$.
- Otherwise shrink or expand L_I as previously described.

Thanks to the previous update, while costly solutions are computed, CASCBO can exploit a cheaper trust region. When costly solutions are evaluated, and if one of them improves the current best solution while being outside the current trust region, then L_I is restored to the state at which the best costly solution was sampled. So, while computing expensive solutions, CASCBO can retrieve cheap and relevant information.

IV. Experimental setup

In this study, six distinct experiments were conducted. Two of them classify the Poisson Encoded MNIST [16]. The first experiment optimizes a search space of 22 HPs applied globally to the network. While the second experiment optimizes 46 HPs applied layer-wise. The same networks and search spaces were applied to the NMNIST dataset [31].

For the first four experiments, the network architecture is a convolutional spiking neural networks, taken from the original SLAYER paper [15]. Instead of a Spike Response Model (SRM) neuron's model, we used a Leaky Integrate and Fire (LIF) with adaptive threshold. Concerning experiments 5 and 6 applied to the SHD [32] dataset, we optimized an existing architecture [33] using SLAYER, consisting of 3 hidden feed-forward layers. Experiment 5 optimizes 21 HPs applied globally, and the 42 HPs of experiment 6 are applied, when possible, layer-wise. All experiments and HPs are summed-up in Table I. For concision, all boundaries can be found within the given code[2], as well as all experimental data. Continuous HPs, such as the neuron's threshold, are not discretized to keep a wider range of possibilities, including potential silent networks.

Batches from the datasets have the following shape : $B.F.C.H.W$, i.e. batch size, frames, channels, height, and width. The batch size and number of frames are HPs to be tuned. The MNIST, NMNIST and SHD datasets follow the same pattern, respectively $(B.F.1.28.28)$, $(B.F.2.34.34)$ and $(B.F.1.700)$. The Tonic [34] Python package was used to load and convert DVS data into frames. Both MNIST and NMNIST have the same number of samples. They are divided into a training ($\mathcal{D}_{\text{train}}$), validation ($\mathcal{D}_{\text{valid}}$) and test ($\mathcal{D}_{\text{test}}$), sets of respective sizes 48000, 12000 and 10000. The SHD dataset contains fewer samples than previous datasets, but was still divided into three subsets of sizes 6524, 1632 and 2264. All validation sets are randomly extracted from the training sets while keeping classes proportions. A cluster of GPUs from Grid5000 [13] was used for computations. A total of 16 NVIDIA A-100 (40 Gib) were used, one of them was dedicated to the computation of CASCBO. Each node contains 4 GPUs, a 32-cores AMD EPYC 7513 (Zen 3) CPU and 512 GiB of RAM. CASCBO was parallelized using OpenMPI, instantiated with `Zellij`[3] and `BoTorch` [35]. The LAVA-DL[4] simulator is used to implement SLAYER.

[2]https://github.com/ThomasFirmin/fidelity_hpo_snn

V. Results and discussion

A. Analysis of the best solutions

Results presented in Table I are selected according to the best accuracy on $\mathcal{D}_{\text{valid}}$ found by CASCBO, and final accuracies are computed on $\mathcal{D}_{\text{test}}$. The budget in the seventh column is in GPU hours. So, to get the real duration of an experiment, in hours, the budget should be divided by the number of used GPUs, here 16. To evaluate stochasticity of the best solutions, SNNs are retrained 16 times during 25 or 100 epochs, depending on potential overfitting. To our knowledge, in the literature of HPO applied to SNNs, we optimized the highest number of hyperparameters while maintaining competitive accuracies on standard benchmarks. Although experiments 2, 4 and 6 are more difficult due to the higher number of HPs, this does not always translate into lower accuracies. In Table I, one can even observe improvements when HPs are optimized layer-wise. However, it is not always the case, depending on the parametrization of CASCBO, available resources and budget.

Concerning experiments 1 to 4, and compared to the baseline results from [15], we can notice that our results, are more stochastic, even if they are close to the baseline, and slightly better for experiment 4. We assume this stochasticity might be explained by the usage of an adaptive LIF, rather than SRM neurons. An additional source of uncertainty may arise from the difference in simulators used. The baseline was computed using the original SLAYER simulator[5], whereas we utilized SLAYER-2 from LAVA-DL.

Moreover, our networks are trained on fewer data than the given baseline. Indeed, a good practice [10], [12] when doing HPO, even manually, is to have three subsets of data, $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{valid}}$ and $\mathcal{D}_{\text{test}}$, to prevent overfitting the HPs, and to guarantee a certain degree of generalizability. When doing HPO, a bias is introduced because of an additional step in the design of a SNN.

Concerning experiments 5 and 6, results indicate that by tuning HPs, one could significantly improve the accuracies of handcrafted architecture presented in [33].

In Fig. 1, we used the t-SNE algorithm [36] to apply dimensionality reduction on evaluated solutions produced by CASCBO. One can see there exist different areas (clusters of points) of the search space containing good solutions. The darkest and biggest clusters, in all six figures, indicate groups of solutions that are mostly silent networks. So, we can deduce that some part of the search space, distinct from the ones containing high accuracy solutions, contains many infeasible solutions, i.e. silent networks.

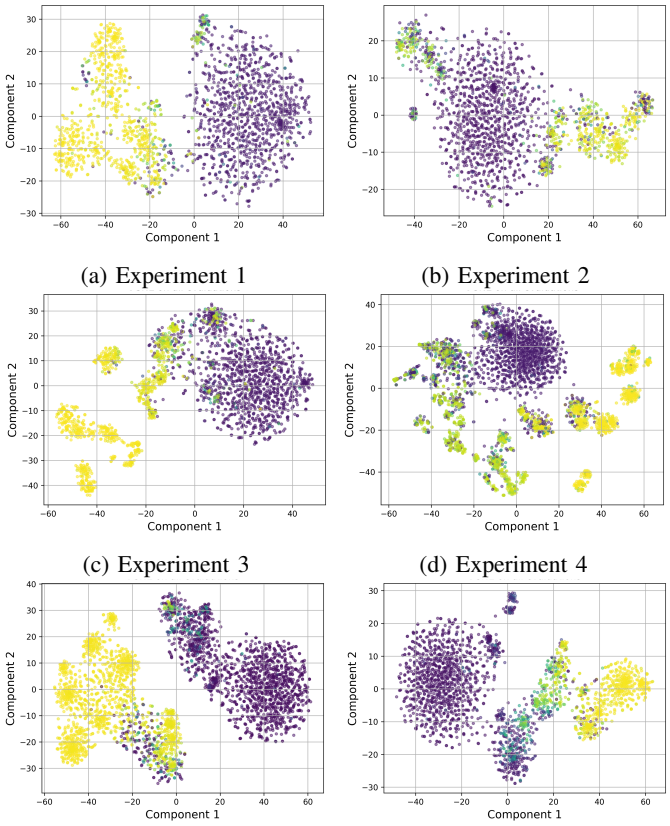
[3]https://github.com/ThomasFirmin/zellij

[4]https://github.com/lava-nc/lava-dl

[5]https://bitbucket.org/bamsumit/slayer

TABLE I: Summary of experiments

#	Dataset	HPs	Selected HPs		α, β	Budget	Test accuracy	Source
1	MNIST	22	λ^{train}, frames, epochs, batch size, output spike rate for true and false label, learning rate, time constant of spike function derivative, gradient scale, threshold, threshold step, current decay, voltage decay, threshold decay, refractory decay, neurons dropout	momentum of SGD, learning rate schedule, filters and kernel size for each conv layer	$5, 3\%$	224	97.56 ± 0.70	99.36 ± 0.05 [15]
2	MNIST	46			$5, 3\%$	224	99.06 ± 0.14	
3	NMNIST	22			$5, 3\%$	640	94.96 ± 2.26	99.20 ± 0.02 [15]
4	NMNIST	46			$5, 3\%$	640	98.85 ± 0.38	
5	SHD	21		both momentums of ADAM, number of neurons for each dense layer	$10, 1\%$	224	89.55 ± 1.74	70.58 ± 1.9 [33]
6	SHD	42			$10, 1\%$	224	90.19 ± 2.43	



(a) Experiment 1 (b) Experiment 2

(c) Experiment 3 (d) Experiment 4

(e) Experiment 5 (f) Experiment 6

Fig. 1: t-SNE applied on all evaluated solutions returned by CASCBO. The lighter or yellower the point, the higher the accuracy.

Concerning λ^{train}, it is not always necessary to use the full training dataset to obtain good accuracies. With proper parameter tuning, on MNIST, we can reach about 96% of accuracy by only using 10% of $\mathcal{D}_{\text{train}}$. Most of the dataset is then used to refine the accuracy to reach about 99% validation accuracy. Similar behaviors can be observed on NMNIST and SHD.

B. Analysis of CASCBO

In this section, we analyze the behaviors of CASCBO during the optimization process. Table II sums-up the best solutions found by CASCBO, and selected according to the best accuracy on $\mathcal{D}_{\text{valid}}$. The number of evaluated HPs combinations, the proportion of sampled silent networks and their influence on the budget (GPU hours) are shown in the three last columns.

TABLE II: Results from CASCBO without retraining

#	c_{out}	$Cost$	Train	Valid	# eval.	% silent	% budg.
1	-0.03	1228	97.31	97.54	2073	55.3	2.2
2	-0.03	1412	98.55	98.28	1823	70.5	1.6
3	-0.03	4255	97.34	97.05	2291	57.0	0.6
4	-0.03	1523	98.65	98.33	3666	55.9	4.0
5	-0.10	437	94.96	98.77	3469	45.5	1.6
6	-0.10	1029	99.83	98.77	2367	49.2	2.1

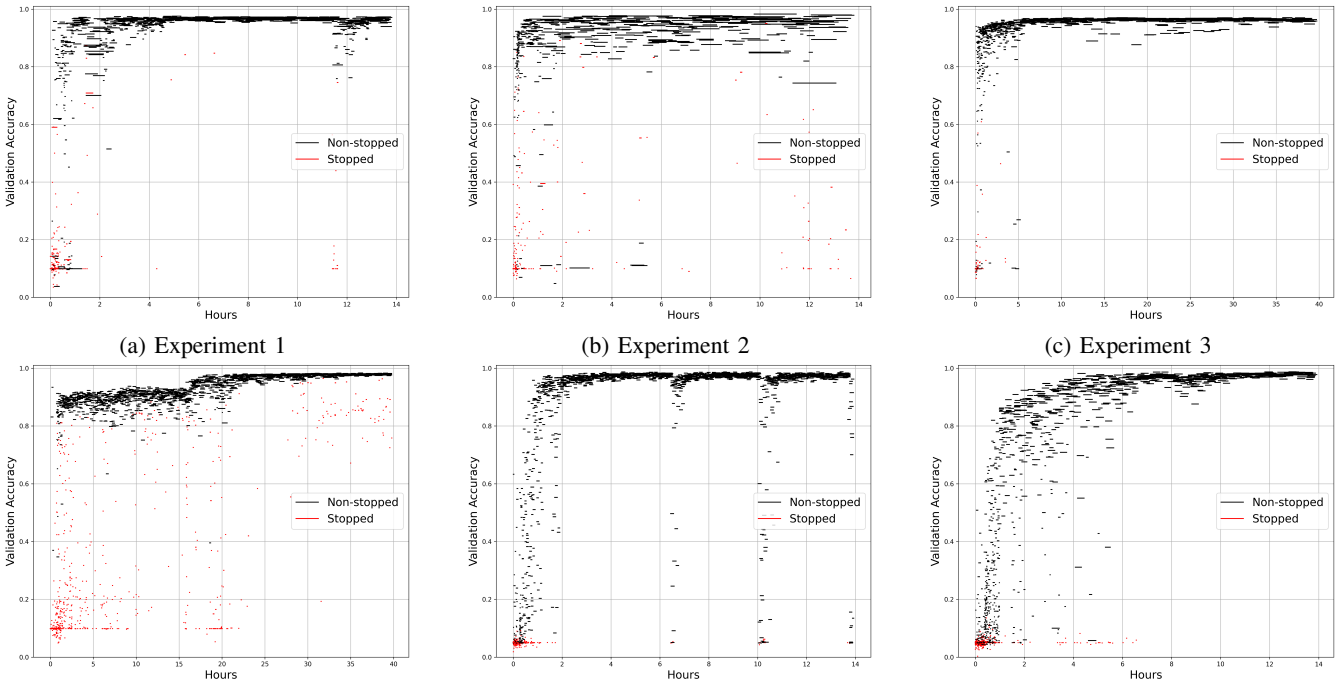
One can see that while almost half of the sampled solutions, 70% for experiment 2, are silent networks, their impact on the budget is greatly diminished, from 0.6% up to only 4%. Additionally, Fig. 2 illustrates that most of the silent networks are sampled at the beginning of the optimization (red lines) when λ^{train} and the cost are low. Then, CASCBO can quickly converge toward fully trained networks (black lines). The differences of $Cost$ and number of evaluated solutions between experiment 3, 4, 5 and 6, is explained by a focus of CASCBO on solutions with a lower or higher number of epochs.

So, multi-fidelity, combined with constraints and early stopping, allows to quickly escape from areas where silent networks are located. These areas are represented by the biggest and darkest clusters in Fig. 1.

The results presented in Fig. 3 serve as a baseline comparison with [5]. By using the same number of GPUs, we were able to obtain a faster convergence and similar results on Poisson encoded MNIST. The SCBO algorithm, with constraints on the spiking activity, required at least 900 GPU hours before ending the first exploitation phase (after about 60 hours), while in CASCBO this phase ended after 180 GPU hours (12 hours). Similar behaviors are observed during the first exploration phase. In [5], this phase ended after about 525 GPU hours, while CASCBO required about 60 GPU hours. So, including fidelity parameters and costs within the HPO of SNNs, allows to significantly increase the convergence of the algorithm while maintaining competitive performances.

VI. Conclusion

In this paper, we leveraged multi-fidelity optimization to handle HPs influencing the cost of training a SNN. The expensiveness of SNNs can be explained by diverse HPs, such



(a) Experiment 1 (b) Experiment 2 (c) Experiment 3

(d) Experiment 4 (e) Experiment 5 (f) Experiment 6

Fig. 2: Validation accuracies according to start and end training dates of all evaluated solutions. A horizontal line represents the training period of a single SNN. The longer the line, the more expensive the training. A red line represents a silent network that has been stopped, and a black line is a fully trained SNN.

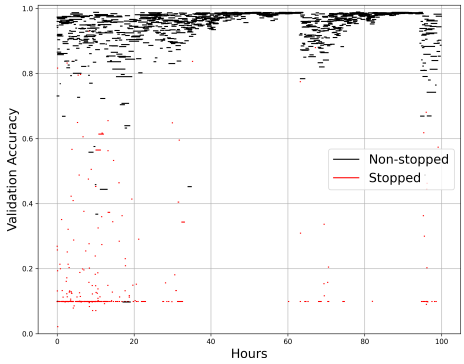


Fig. 3: Validation accuracies optimizing 20 HPs with SCBO during 1600 GPU hours and on MNIST (Train: 98.91 ± 0.12, Valid: 98.57 ± 0.25, Test: 98.80 ± 0.19) [5].

as the temporal resolution of the simulator, the size of the network or the size of the training dataset. We designed a new Bayesian optimization approach called CASCBO, based on SCBO [11], allowing to consider costs of evaluations, constraints, and high dimensional search spaces. To improve the detection of *silent networks* and reduce their negative influence on the budget, we redefined specific constraints. These are based on the number of samples that output at least; and not at most, α spikes for a certain layer. This new definition follows a hypergeometric law, giving a stronger quantitative and probabilistic meaning to the constraints. Moreover, to handle asynchronicity and the high stochasticity of a SNN evaluation, we propose an asynchronous trust region with back-in-time, providing the opportunity to CASCBO to restore a previous trust region if a better and late evaluation ends outside the current trust region.

Experiments indicate that CASCBO can optimize high dimensional and constrained search spaces, including fidelity HPs. These HPs have an effect on the training time of SNNs, and so on the budget allocated to silent networks. The proposed methodology makes the impact of silent networks negligible, allowing to define a more general and flexible search space. The optimization process can quickly stabilize the high sensitivity of SNNs to their HPs, before tackling expensive evaluations. Our approach empirically indicates that expensive evaluation on the full training dataset is not necessary to obtain high accuracies. Despite the complexity of the problem, the presented results remain competitive compared to their baseline accuracies, yet better. Additionally, when doing HPO, to prevent overfitting the HPs [12] a more rigorous approach is necessary. It includes train, validation, and test subsets; meaning that we train on fewer data compared to the baselines, which improves the generalizability of our solutions. Finally, we are working on applying our methodology to cost-aware multi-objective HPO of SNNs where minimizing the number of spikes is important, while maintaining a minimum spiking activity.

ACKNOWLEDGMENTS

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

This work is supported by a public grant overseen by the French National Research Agency (ANR) as part of the 'PEPR IA France 2030' programme (Emergences project ANR-23-PEIA-0002). It is also supported by IRCICA(CNRS and Univ. Lille USR-3380) and by the University of Lille, the ANR-20-THIA-0014 program AI_PhDLille and the ANR PEPR AI and Numpex. E-G. T. was partially supported by the EXAMA (Methods and Algorithms at Exascale) project under grant ANR-22-EXNU-0002.

REFERENCES

[1] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997.

[2] A. Taherkhani, A. Belatreche, Y. Li, G. Cosma, L. P. Maguire, and T. McGinnity, "A review of learning in biologically plausible spiking neural networks," *Neural Networks*, vol. 122, pp. 253–272, Feb. 2020.

[3] E. Stromatias, D. Neil, F. Galluppi, M. Pfeiffer, S.-C. Liu, and S. Furber, "Scalable energy-efficient, low-latency implementations of trained spiking Deep Belief Networks on SpiNNaker," in *2015 International Joint Conference on Neural Networks (IJCNN)*. Killarney, Ireland: IEEE, Jul. 2015, pp. 1–8.

[4] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber, "SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, Aug. 2013.

[5] T. Firmin, P. Boulet, and E.-G. Talbi, "Parallel Hyperparameter Optimization Of Spiking Neural Networks," Nov. 2023, working paper or preprint.

[6] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*, 1st ed. Cambridge University Press, Aug. 2002.

[7] W. Guo, H. E. Yantir, M. E. Fouda, A. M. Eltawil, and K. N. Salama, "Toward the Optimal Design and FPGA Implementation of Spiking Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 8, pp. 3988–4002, Aug. 2022.

[8] M. Parsa, J. P. Mitchell, C. D. Schuman, R. M. Patton, T. E. Potok, and K. Roy, "Bayesian-based Hyperparameter Optimization for Spiking Neuromorphic Systems," in *2019 IEEE International Conference on Big Data (Big Data)*. Los Angeles, CA, USA: IEEE, Dec. 2019, pp. 4472–4478.

[9] M. Parsa, C. Schuman, N. Rathi, A. Ziabari, D. Rose, J. P. Mitchell, J. T. Johnston, B. Kay, S. Young, and K. Roy, "Accurate and Accelerated Neuromorphic Network Design Leveraging A Bayesian Hyperparameter Pareto Optimization Approach," in *International Conference on Neuromorphic Systems 2021*. Knoxville TN USA: ACM, Jul. 2021, pp. 1–8.

[10] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A. Boulesteix, D. Deng, and M. Lindauer, "Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges," *WIREs Data Mining and Knowledge Discovery*, vol. 13, no. 2, p. e1484, Mar. 2023.

[11] D. Eriksson and M. Poloczek, "Scalable Constrained Bayesian Optimization," Feb. 2021, arXiv:2002.08526 [cs, stat].

[12] M. Hosseini, M. Powell, J. Collins, C. Callahan-Flintoft, W. Jones, H. Bowman, and B. Wyble, "I tried a bunch of things: The dangers of unexpected overfitting in classification of brain data," *Neuroscience & Biobehavioral Reviews*, vol. 119, pp. 456–467, Dec. 2020.

[13] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard, "Grid'5000: a large scale and highly reconfigurable grid experimental testbed," in *The 6th IEEE/ACM International Workshop on Grid Computing, 2005.* Seattle, WA, USA: IEEE, 2005, p. 8 pp.

[14] A. Vigneron and J. Martinet, "A critical survey of STDP in Spiking Neural Networks for Pattern Recognition," in *2020 International Joint Conference on Neural Networks (IJCNN)*. Glasgow, United Kingdom: IEEE, Jul. 2020, pp. 1–9.

[15] S. B. Shrestha and G. Orchard, "SLAYER: Spike Layer Error Reassignment in Time," *arXiv:1810.08646 [cs, stat]*, Sep. 2018, arXiv: 1810.08646.

[16] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[17] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha, "A Low Power, Fully Event-Based Gesture Recognition System," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, Jul. 2017, pp. 7388–7397.

[18] H. Zheng, Y. Wu, L. Deng, Y. Hu, and G. Li, "Going Deeper With Directly-Trained Larger Spiking Neural Networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, pp. 11 062–11 070, May 2021.

[19] Y. Kim and P. Panda, "Optimizing Deeper Spiking Neural Networks for Dynamic Vision Sensing," *Neural Networks*, vol. 144, pp. 686–698, Dec. 2021.

[20] A. Vicente-Sola, D. L. Manna, P. Kirkland, G. Di Caterina, and T. Bihl, "Keys to accurate feature extraction using residual spiking neural networks," *Neuromorphic Computing and Engineering*, vol. 2, no. 4, p. 044001, Dec. 2022.

[21] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and Efficient Hyperparameter Optimization at Scale," Jul. 2018, arXiv:1807.01774 [cs, stat].

[22] H. Hazan, D. J. Saunders, H. Khan, D. Patel, D. T. Sanghavi, H. T. Siegelmann, and R. Kozma, "BindsNET: A Machine Learning-Oriented Spiking Neural Networks Library in Python," *Frontiers in Neuroinformatics*, vol. 12, p. 89, Dec. 2018.

[23] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator," *eLife*, vol. 8, p. e47314, Aug. 2019.

[24] S. R. Kulkarni and B. Rajendran, "Spiking neural networks for handwritten digit recognition—Supervised learning and network optimization," *Neural Networks*, vol. 103, pp. 118–127, Jul. 2018.

[25] Y. Li, T. Geller, Y. Kim, and P. Panda, "Seenn: Towards temporal spiking early-exit neural networks," 2023.

[26] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," *arXiv:1206.2944 [cs, stat]*, Aug. 2012, arXiv: 1206.2944.

[27] K. Kandasamy, A. Krishnamurthy, J. Schneider, and B. Poczos, "Parallelised bayesian optimisation via thompson sampling," in *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Storkey and F. Perez-Cruz, Eds., vol. 84. PMLR, 09–11 Apr 2018, pp. 133–142.

[28] J. P. Folch, R. M. Lee, B. Shafei, D. Walz, C. Tsay, M. van der Wilk, and R. Misener, "Combining Multi-Fidelity Modelling and Asynchronous Batch Bayesian Optimization," Feb. 2023, arXiv:2211.06149 [cs, stat].

[29] D. Eriksson, M. Pearce, J. R. Gardner, R. Turner, and M. Poloczek, "Scalable Global Optimization via Local Bayesian Optimization," Feb. 2020, arXiv:1910.01739 [cs, stat].

[30] E. H. Lee, V. Perrone, C. Archambeau, and M. Seeger, "Cost-aware Bayesian Optimization," Mar. 2020, arXiv:2003.10870 [cs, stat].

[31] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades," *Frontiers in Neuroscience*, vol. 9, Nov. 2015.

[32] B. Cramer, Y. Stradmann, J. Schemmel, and F. Zenke, "The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 7, pp. 2744–2757, Jul. 2022.

[33] F. C. Bauer, G. Lenz, S. Haghighatshoar, and S. Sheik, "EXODUS: Stable and efficient training of spiking neural networks," *Frontiers in Neuroscience*, vol. 17, p. 1110444, Feb. 2023.

[34] G. Lenz, K. Chaney, S. B. Shrestha, O. Oubari, S. Picaud, and G. Zarrella, "Tonic: event-based datasets and transformations." Jul. 2021.

[35] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy, "Botorch: a framework for efficient monte-carlo bayesian optimization," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20. Red Hook, NY, USA: Curran Associates Inc., 2020, p. 15.

[36] L. v. d. Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.