Investigation of Novel SBF-Automata Architecture for Periodicity Finding Solutions at Edge Systems

Michael Joseph Tarlton[1], Anis Yazidi[2], Ali Muhtaroglu[3], and Gustavo Mello[4]

***Abstract*— Periodicity finding is a critical element in many online learning domains such as web crawling, stock market analysis, robotics, and environmental monitoring. The ability to identify, associate, and internally reproduce intervals of time is a fundamental cognitive element involved in all neural functions in the animal model as well, acting as the basis for learning across the brain. The Striatal Beat Frequency (SBF) model is a well-supported neuroscientific representation of time-intervals within a biological neural architecture and periodicity learning. We transpose the conceptual framework of the SBF into a biologically inspired adaptation, named SBF-Automata (SBF-A): a reinforcement learning (RL) based framework aimed at addressing the periodicity finding problem in real-time and online artificial systems. Initial analysis and simulations indicate that SBF-A provides an attractive alternative to real-time periodicity finding problems at the edge, for which low implementation complexity is of paramount importance.**

I. Introduction

Time-based learning is a critically underdeveloped area in artificial neural networks (ANNs). Moreover, it is virtually absent in online and deep-learning models, which lack the capability to learn temporal associations in real-time edge deployments. Paradoxially, biological neuronal networks are inherently time-based, as evidenced by mechanisms ranging from the firing intervals of spiking communication and spike-timing-dependent plasticity to the larger scale temporal coordination of regional and global brain oscillations [1]. Neural dynamics operate across a wide range of time scales, from milliseconds to the representation of time over seconds, minutes, and beyond [2].

Identifiying time intervals between significant temporal points, or detecting the regular or semi-regular occurrence of events over time, constitutes the *periodicity finding problem*, a common challenge in both natural and artificial systems. In animals, this problem corresponds to *Interval Timing (IT)*: the brain's capacity to track and associate external events over specific time intervals. Within the field of neuroscientific reinforcement learning (RL), IT is typically studied using the ***Fixed-Interval (FI) task*** [3]. In this task, a subject is conditioned to learn a specific time interval by associating a stimulus with a rewarded action. Each trial begins with a stimulus, and once the target time interval - referred to as the **criterion time** T_C - has elapsed, a reward becomes available, contingent on a particular action (e.g. pressing a lever.) For humans, perceivable time intervals generally range from sub-second to supra-second scales ($\sim 300 - 1000ms$) [4]. Investigating how $\sim 10ms$ neuronal dynamics scale to enable IT could inspire novel and improved designs for ANNs.

Associating events separated in time represents a fundamental challenge for systems operating in dynamic and time-sensitive environments. Periodicity finding problems, such as those encountered in web crawling, online learning, feedback control, and environmental monitoring, depend on the prediction of temporal properties within otherwise noisy data streams. Current approaches to periodicity finding often rely on computationally complex methods, which may be incompatible with the low-energy, real-time processing demands of edge hardware [5]. Traditional neuroscientific models of IT are influenced by *Von Neumann* computing paradigm, wherein dedicated neural modules are required for clocking and storing memorized time intervals. However, such models remain poorly defined at the level of neuronal architecture [6].

The Striatal Beat Frequency (SBF) model is a neuroscientific model of IT and periodic activity reproduction in mammalian brains [7][8], which offers features appealing to machine learning (ML): decentralized & asynchronous temporal processing, leveraging of existing neural mechanisms, and integration of activity from assemblies handling other stimulus aspects [9]. The SBF model incorporates well-established neural circuits for motor control, execution, reinforcement, and reward, with its neurobiological architecture strongly validated through simulation [10]. The model consists of endogenous *oscillators* with diverse frequencies, driven by individual neuronal firing rates or population activity. Oscillatory pulses are sent to downstream "coincidence detectors," which associate these pulses with meaningful stimuli. Upon top-down reinforcement (e.g. a salient event or reward), the phase pattern of synchronized oscillatory activity is encoded in the weights of a neuronal ensemble [11].

As the SBF model encodes internal time information on the distribution of weights in a minimal neural network, it is an ideal candidate for ANN applications. The biologically inspired method of learning and modifying weights may provide a novel low-complexity and low-energy alternative

[1]Michael Tarlton is a PhD student at the Faculty of Technology, Art, and Design: Artificial Intelligence (AI) Lab, Oslo Metropolitan University, Pilestredet 46, 0167, Oslo, Norway michaelt@oslomet.no

[2]Prof. Dr. Anis Yazidi is at the Dept. of Computer Science and is a member of AI Lab, Faculty of Technology, Art, and Design, Oslo Metropolitan University, Pilestredet 46, 0167, Oslo, Norway anisy@oslomet.no

[3]Assoc. Prof. Dr. Ali Muhtaroglu is at Dept. of Mechanical, Electrical and Chemical Eng. and is a member of ADEPT research group and AI Lab, Faculty of Technology, Art, and Design, Oslo Metropolitan University, Pilestredet 46, 0167, Oslo, Norway alimuhta@oslomet.no

[4]Assoc. Prof. Dr. Gustavo Mello is at the Dept. of Computer Science and is a member of AI Lab, Faculty of Technology, Art, and Design, Oslo Metropolitan University, Pilestredet 46, 0167, Oslo, Norway gustavom@oslomet.no

to traditional methods utilized in deep neural networks. Although SBF is a well-supported neuroscientific model of RL in animal behavior with implementations in simulation [10], it has not yet been introduced to a ML context.

We propose SBF-Automata (SBF-A) in this work: an adaptation of the SBF model into a RL framework for use in continuous activation ANNs in resource constrained real-time edge applications. As the SBF-A follows a neuronal spiking paradigm, the model is suitable for implementation in ultra-low-power and asynchronous hardware, such as autonomous robotics and edge computing devices deployed in resource constrained environments [12][13]. Unlike traditional RL models which rely on tabular data storage, The SBF-A encodes learned policies directly into the weights of the network. This differs from Deep-RL methods [14], as Deep-RL networks are incapable of online learning, and rely on large sample batches over multiple epochs. Unlike back-propagation, SBF-A applies weight updates locally, allowing for online, in-situ, and few-shot learning. The original contributions of the work can be summarized as follows:

(i) the SBF-A: a novel automata model with oscillator and executive units with potential scalability to large networks
(ii) a family of weight update algorithms for the SBF-A, capable of learning and reproducing periodic activity
(iii) comparison of the computational complexity, learning performance and accuracy of the SBF-A model against standard periodicity finding approaches,
(iv) comparison across alternative SBF-A weight distribution algorithms using metrics that provide insights into differences in delay, accuracy, activity and energy consumption.

In the following section, we outline how the basis of the SBF model can be fitted to a learning automata context in a novel neural model of learning: the SBF-A. We establish the general framework of the SBF-A, as well as the family of weight update algorithms that have been successfully deployed. In sections IV and V we draw comparisons between the SBF-A and established periodicity finding methods, and outline the experiments applied to evaluate the SBF-A model, respectively. The results and discussion in section VI are followed by conclusions from this work in section VII.

II. The SBF-Automata

SBF-A is a naive RL model based on SBF with two main parts: 1. the **oscillator block**, containing a set of oscillator units, each of which "peaks" in activity at a unique periodicity; and 2. the **executive unit**, which integrates the individual unit activity from the oscillator block and decides whether or not to perform an action.

Oscillator units act discretely, mimicking the tonic firing of a single neuron. An individual unit has a unique periodicity θ i.e. it activates on time-steps coinciding with its period. For example, unit i with $\theta_i = 3$ has activity a_i=1 during time-step t=3, 6, 9, ... (Fig. 1). Each oscillator is weighted. Initial weights are evenly distributed, such that initial weights are w_i=$1/N_{osc}$, where N_{osc} is the total number of oscillator

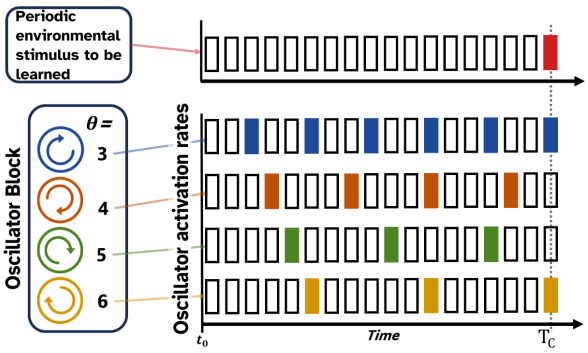

Fig. 1: Discrete oscillator unit activity over the time-space.

units in the set. A special oscillator with zero-periodicity, termed the *no-action oscillator*, is also included in this set. This unit votes for no-action at every time-step, acting as a broad inhibitory signal, tempering hyperactivity by absorbing excess weight in the system.

At the start of a new trial all oscillator phases are reset to their initial state. Oscillators then run continuously until a reset signal is received, denoting the start of a new trial. At each time-step t, activity from oscillator units whose periods exactly divide the current time-step is projected to the executive unit. The executive unit then makes a Markov decision based on the weighted and summed activity. If an action is taken by the SBF-A, the environment then elicits a response (reward, no-reward) conditional on if the action has been performed on a valid time-step. Upon receiving environmental stimulus, the SBF-A updates its weights and learning parameters according to the chosen algorithm, taking into account the reward signal received from the environment $r(t)$=$[0, 1]$. After some period of training, the distribution of weight should correspond to a distribution of oscillatory signals which best inform the executive unit to act on the correct time-steps.

A. Model Initialization

Each oscillator i is assigned a unique periodicity θ_i, selected from a uniformly distributed range to ensure diverse activation patterns. Initial weights $w_i(0)$ are assigned uniformly across all oscillators: $w_i(0) = 1/N_{\text{osc}}$.

B. Operational Stages

The SBF-Automata operates in two distinct phases at each time-step:

1) **Activiation Stage**:
- Determine the set of active oscillators $\mathcal{A}(t)$, where $\mathcal{A}(t) = \{i : a_i(t) = 1\}$.
- An oscillator unit is defined as active if its cyclic period is in phase with the current time-step. If $t \ mod \ \theta_i$=0 then $a_i(t)$=1, else $a_i(t)$=0.
- Calculate the activation rate $p(t)$ as the sum of weights of active oscillators: $p(t) = \sum_{i \in \mathcal{A}(t)} w_i(t)$.
- Assess whether action occurs by comparing $p(t)$ against a random number x uniformly distributed between zero to one: $\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$, such that probability of action corresponds to $p(t)$.

2) **Selection Stage**:
   - If action occurs ($p(t) \geq x$):
     - Observe the reward signal $r(t)$.
     - Apply weight update rules for active and inactive oscillators based on the observed reward.
   - If action does not occur ($p(t) < x$):
     - No action is taken; weights remain unchanged.

The reader may refer to Algo. 1 for algorithmic format.

Algorithm 1 SBF-A General Framework

```
Initialize oscillator weights w_i(0) = 1/N_osc for all i
for each time-step t from 1 to T do
    Determine active oscillators A(t) based on periodicity and current time-step
    Compute activation rate p(t) = Σ_{i∈A(t)} w_i(t)
    if p(t) < x then ▷ action condition based on threshold x
        Observe reward r(t)
        Update oscillator weights for active and inactive oscillators ▷ Refer to specific algorithm for update rules
    else
        No action
    end if
end for
```

III. SBF-A Algorithms

Four different SBF-A algorithms are investigated as part of this work, each differ in how oscillator weights are redistributed based on state-action-reward feedback. The choice of algorithm depends on the desired sensitivity to reward magnitude and the specific learning dynamics required for the chosen task. The general weight update procedure is outlined next, followed by details of different algorithms.

General Update Procedure

1) **Compute Activation Metrics**:
   a) $W_{\text{active}}(t)$: Total weight of active oscillators
   b) $W_{\text{inactive}}(t)$: Total weight of inactive oscillators
   c) $N_{\text{active}}(t)$: Number of active oscillators
   d) $N_{\text{inactive}}(t)$: Number of inactive oscillators
2) **Apply Update Rules**: Depending on the observed reward $r(t)$ and the specific algorithm, adjust the weights $w_i(t)$ accordingly.

A. Uniform Redistribution - Magnitude Agnostic (UR-MA)

The Weighted Majority Algorithm (WMA) [15] is the basis for the UR-MA weight redistribution algorithm. The algorithm is Magnitude Agnostic (MA) in that the environmental reward is interpreted as a boolean, ignoring any reward value magnitude. In any reward-state the weight is redistributed from oscillator units which are in "incorrect" state, i.e. when no reward is retrieved, weight is redistributed from active units to inactive units, and when reward *is* retrieved, weight is redistributed from inactive units to active units. We uniformly apply this redistribution by summing the weights of units in the incorrect state, adjusting the amount scaled by the learning parameters (α, β), and evenly applying the sum of that weight to units in the "correct" state, while reductively scaling the "incorrect" unit weight. The **UR-MA** is given by: Reward ($r(t) > 0$):

$$w_i(t+1) = \begin{cases} w_i(t) + \frac{W_{\text{inactive}}(t) \cdot \alpha}{N_{\text{active}}(t)}, & \text{if active,} \\ w_i(t) \cdot (1-\alpha), & \text{if inactive.} \end{cases} \tag{1}$$

Punishment ($r(t) = 0$):

$$w_i(t+1) = \begin{cases} w_i(t) \cdot (1-\beta), & \text{if active,} \\ w_i(t) + \frac{W_{\text{active}}(t) \cdot \beta}{N_{\text{inactive}}(t)}, & \text{if inactive.} \end{cases} \tag{2}$$

B. Uniform Redistribution - Magnitude Sensitive (UR-MS)

The UR-MA is extended to the Magnitude Sensitive (MS) variant, the UR-MS. The UR-MS recognizes and integrates the magnitude of the reward signal r=$[0, 1]$, eliminating separate handling of reward and punishment cases, and redistributing weight proportionally with actual amount of reward retrieved. This allows the redistribution algorithm to account for how "close" oscillator units were to the target. This is useful in cases where a freshness measure is applied to reward. The **UR-MS** is given by:

$$w_i(t+1) = \begin{cases} w_i(t) + r(t) \cdot \dfrac{W_{\text{inactive}}(t) \cdot \alpha}{N_{\text{active}}(t)} \\ \quad -(1-r(t)) \cdot w_i(t) \cdot \beta, & \text{if active,} \\ w_i(t) \cdot (1 - r(t) \cdot \alpha) \\ \quad +(1-r(t)) \cdot \dfrac{W_{\text{active}}(t) \cdot \beta}{N_{\text{inactive}}(t)}, & \text{if inactive.} \end{cases} \tag{3}$$

C. Proportional Contribution Redistribution (PCR)

Uniform Redistribution algorithm is further enhanced into Proportional Contribution Redistribution (PCR). Taking inspiration from Long Term Potentiation (LTP) learning in the neuronal model [16], the weights are adjusted based on the individual unit contribution relative to the total oscillator activity. Hence, the weight redistribution is scaled proportional to the weighted sum of activity. The **PCR** formula is:

$$c_i(t) = \begin{cases} \frac{w_i(t)}{W_{\text{active}}(t)}, & \text{if active,} \\ \\ \frac{w_i(t)}{W_{\text{inactive}}(t)}, & \text{if inactive,} \end{cases} \tag{4}$$

where $c_i(t)$ is the relative contribution of unit i at time-step t. The PCR for the MA regime (**PCR-MA**) is defined as,

$$w_i(t+1) = \begin{cases} w_i(t) + c_i(t) \cdot W_{\text{inactive}}(t) \cdot \alpha, \\ \qquad \text{if } r(t) > 0 \text{ and active,} \\ w_i(t) \cdot (1-\alpha), \\ \qquad \text{if } r(t) > 0 \text{ and inactive,} \\ w_i(t) + c_i(t) \cdot W_{\text{active}}(t) \cdot \beta, \\ \qquad \text{if } r(t) = 0 \text{ and inactive,} \\ w_i(t) \cdot (1-\beta), \\ \qquad \text{if } r(t) = 0 \text{ and active,} \end{cases} \tag{5}$$

and is expanded to the MS regime (**PCR-MS**) as,

$$w_i(t+1) = \begin{cases} w_i(t) + r(t) \cdot c_i(t) \cdot W_{\text{inactive}}(t) \cdot \alpha \\ \qquad -(1-r(t)) \cdot w_i(t) \cdot \beta, \text{ if active,} \\ (1-r(t)) \cdot c_i(t) \cdot W_{\text{active}}(t) \cdot \beta \\ \qquad +w_i(t) \cdot (1-r(t) \cdot \alpha), \text{ if inactive.} \end{cases} \tag{6}$$

This brings the algorithm more in-line with a Dopaminergic LTP learning rule [16], where strength of the reward attenuates the learning error during weight updates.

IV. Computational Complexity

A. Complexity Per Update Step in SBF-A

Unlike traditional time-stepped algorithms, the SBF-A does not necessarily update its weights at *every* time-step. An update occurs only *when the system takes an action*. Determining which oscillators are active and redistributing weights both require at most $O(N)$ work, where N is the number of oscillators:

1) **Check Active Oscillators**: For each oscillator, we verify whether its period divides the current time index. This is a linear pass across N oscillators.
2) **Weight Update**: Whether we use PCR, or UR rules, each update step loops over the N oscillators to adjust weights.

Hence, *each update step* takes $O(N)$ time, even if multiple oscillators happen to be active at once.

B. Contextual Bandit Complexity in Comparison

A *contextual bandit* perspective typically entails more expensive update steps. For example, LinUCB [17] requires inverting (or rank-1 updating) a $N \times N$ matrix, where N is the number of features. Naive inversion costs $O(N^3)$ per update. Even if updates do not happen every step, *any* LinUCB update must handle matrix operations of $O(N^3)$ complexity, significantly higher than the $O(N)$ in SBF-A.

C. Implications for Scalability and Biological Inspiration

Since each SBF-A update step is $O(N)$, it scales favorably with the number of oscillators. By contrast, LinUCB's matrix operations inflate to $O(N^3)$ per update step. This divergence becomes critical as N grows. Furthermore, SBF-A's incremental weight shift reflects the kind of distributed, asynchronous processing observed in real neural circuits. Thus, both from a computational and biological standpoint, SBF-A offers a simpler and more plausible pathway to periodic learning.

V. Methods

A. Experimental Task

We replicate the FI as RL task, where the automata is made to learn a target time interval, T_C, by associating an initial stimulus with a reward. A the beginning of each experiment, the automata is given a start signal to mark a new trial and the start of the interval to be timed. During the interval, no reward is available, and actions taken by the automata elicit no response until the T_C is reached. Reward is available for some window of time after the T_C before the trial is reset. The first action taken by the automata, at or after the time-step corresponding to the T_C, retrieves the reward. The SBF-A resets all oscillators to their initial phase upon obtaining the reward. Likewise, the trial timer resets t=0, and a new task trial begins. This is repeated for some number of trials, over which the automata is expected to arrive at an internal approximation solution for the T_C.

The first environment variant tested is the *Windowed* environment, where the full reward value r=1 is available for $0.1 * T_C$ time-steps after T_C. Following this, a *No Window* environment is also tested, where there is no limitation to how soon the reward can be retrieved. A freshness measure is added for a *Decaying Reward* environment, where reward value is proportional to how quickly it is retrieved by the model, with reward at its maximum value of r=1 at the T_C and decaying at a log-rate with each subsequent time-step, until retrieved by the automata. Decay rate is scaled to the T_C. It is worthwhile to note that decay is ignored in the MA algorithms which interpret any reward as having full value, regardless of the actual value. The last variation is the *Non-Resetting* environment. In this environment, the reward is available with static periodicity, which does not reset to zero on the start of a new trial, and instead is continuously running. The SBF-A oscillators still reset upon retrieval of reward. Reward value decays and is available until retrieval. This emulates classic periodicity finding problems such as optimal web-crawling [18] and provides a challenge to the model as it can no longer rely on an initializing stimulus. Environment variants in the experimental task are summarized in Table I.

B. Experimental Regime

Using a toy model as a proof of concept, the SBF-A's efficacy in identifying and learning a target time interval, is studied. The SBF-A and its update algorithms are capable of converging to the solution T_C when given as a context choice in an oscillator unit's period: $\theta_{solution} = T_C$. The capabilities of the SBF-A in the naive regime, in which no solution oscillators are provided, are examined next. In this regime, learned time-periods must be internally represented in the distribution of weights in the ensemble. It is expected that the discrete SBF-A is limited in performance by the number of oscillator units used. The relatively few oscillator units used in the regime with the largest oscillator set size,

TABLE I: Environment Variants

Environment	Description
Windowed	Full reward value r=1 available for $0.1 * T_C$ time-steps after T_C.
No Window	Full reward value r=1 available for T_C time-steps after T_C.
Decaying Reward	Decaying reward value r=$[0, 1]$ for $r(t)$=$0.99^{(t-T_C)}$.
Non-Resetting	Trials do not restart. Reward is available every T_C time-steps until retrieved or next T_C period begins. Reward decays with same rate as Decaying Reward environment.

N_{osc}<12,000 with uniformly distributed periods and rigidly discrete tonic activity, may show less overlap or coordinated activity in comparison to other simulations [10] which are able to create overlapping receptive fields. Understanding such limitations is an important aspect of the experiment. The relationship between oscillator set density and distribution of oscillator periods with respect to performance, as well as the comparative performance of the update rules are investigated in the naive case. It is expected to observe an increase in model accuracy as oscillator set density (total number of oscillators) increases. All experiments and simulations are carried out in Python; code is available upon request.

C. Periodicity Finding Methods Comparison

The SBF-A is compared against several common time-series periodicity finding methods, including the Fast-Fourier Transform (FFT) [5], the Auto-Correlation Function (ACF) [19], and Autoperiod [20] methods. The above algorithms are tested against the PCR-MS in the toy regime with Decaying Reward and Non-resetting environments. For each trial the PCR-MS activity and reward are recorded as a sparse time-series where: 0 is no action, -1 is miss, and $+1$ is action with reward. At the end of each trial (retrieval of the reward) the resulting time-series is input to the other methods and their resulting period estimate is recorded for that trial. We directly compare the performance of the standard methods agains the SBF-A in fig. 3.

VI. Results

A. Toy Regime

The toy regime is tested for a $T_C = 24$ and a predefined oscillator period distribution set of θ={3, 6, 12, 24, 36, 48}. This regime does not use a no-action oscillator. The SBF-A algorithms successfully maximize the weight of the correct oscillator unit θ=24. The weights are redistributed so that the oscillator with the same period as the T_C obtains the largest weight, while the weights of oscillators whose periods subdivides the T_C have weights distributed to them proportional to how often they correctly contribute, and inversely proportional to how often they incorrectly contribute.

B. Evaluation of SBF-A

The decaying reward value can be observed as a metric of how the SBF-A develops in precision over time. In Fig. 2 the value of the reward recovered is plotted alongside the number of actions taken by the automata agent per trial. All values average from one hundred sessions. The SBF-A maintains a high performance, staying over a mean reward value of R>0.89 and increasing to R∼0.95 after convergence. The average number of actions per trial, starts out relatively low, and decreases further over time to reach stability. While the SBF-A solution effectively reaches convergence at trial 60-73 (Table II), oscillator weights continue to progress and separate until the end of the session.

Fig. 3 illustrates the average delay (total time-steps over the T_C when action is taken) against the periodicity prediction of the other methods. The SBF-A maintains an advantage in stability, speed of learning, and accuracy.

TABLE II: Toy Model Results

Algo	Env	Delay	STD	Lat.	Energy
UR-MA	Window	0.00	0.003	64	0.72
	No Window	0.08	0.025	32	16.61
	Decaying Reward	0.05	0.026	61	9.97
	Non-Resetting	0.12	0.024	23	27.50
UR-MS	Window	0.00	0.002	76	0.46
	No Window	0.06	0.032	35	13.08
	Decaying Reward	0.04	0.021	58	8.34
	Non-Resetting	0.09	0.011	90	19.90
PCR-MS	Window	0.00	0.002	57	0.27
	No Window	0.04	0.022	46	8.07
	Decaying Reward	0.02	0.013	60	4.00
	Non-Resetting	0.07	0.017	61	17.29

Delay is the Peak Delay value, normalized to T_C. "STD" is the Delay Standard Deviation normalized to T_C. "Lat." is Latency. Energy is the total activity of the model multiplied by the normalized Peak Delay value (lower is better).

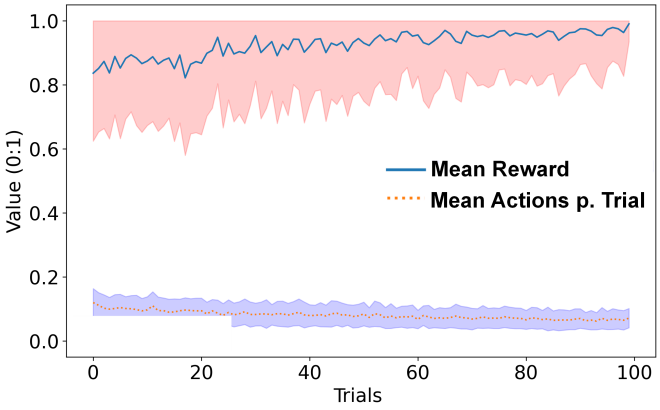

Fig. 2: Mean reward value retrieved and mean actions per trial. Action total is normalized to trial size (T_C). Shaded areas represent the first normal Std Dev.

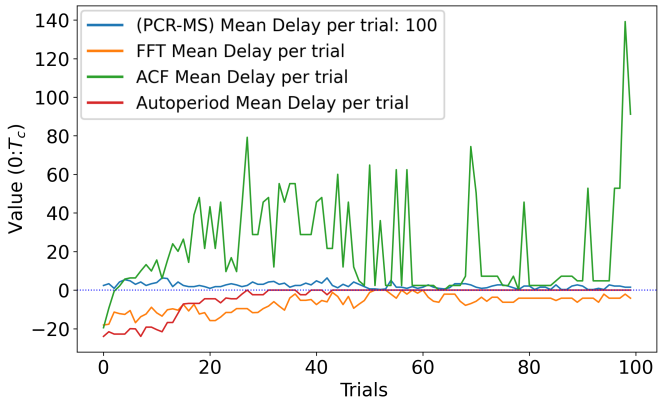

Fig. 3: Comparison of PCR-MS SBF-A against standard periodicity prediction methods. Y-axis is the mean *delay*. X-axis is the trial number in the experiment. Values are averaged over 100 experiments.

C. Comparison of SBF-A Weight Distribution Algorithms

Comparing the performance of different algorithms in Table II, the PCR-MS is the most robust, maintaining the highest accuracy scores in all environments against the UR algorithms. The trade-off is a slight increase in actions

TABLE III: Naive Regime: PCR-MS Decaying Reward

T_C	Dens.	Peak Delay	STD	Lat.	Energy
113	0.1	0.25	0.081	8	98
	0.5	0.11	0.036	3	103
	0.98	0.06	0.022	6	84
1193	0.1	0.07	0.023	4	92
	0.5	0.03	0.010	11	114
	0.98	0.02	0.008	11	99
11939	0.1	0.05	0.015	5	143
	0.5	0.02	0.006	9	169
	0.98	0.01	0.004	7	163

"Dens." is oscillator density. Peak Delay value is normalized to T_C. "STD" is the Delay standard deviation normalized to T_C. "Lat." is Latency. Energy is the total activity of the model multiplied by the normalized Peak Delay value (lower is better).

taken, however, this is negligible in comparison with only a $\sim 2{-}4\%$ increase in total activity over the other algorithms.

D. Naive Regime: Distribution and Density

The true measure of the SBF-A's efficacy, is applying it in a naive regime where a solution is not explicit. Instead, collective oscillator activity is required to compute a solution. PCR-MS is used in this example with a decaying reward environment and three different T_C resolutions: $T_C{=}\{113, 1193, 11939\}$ We use prime numbers to avoid the simplicity of sub-divisor oscillator periods. The oscillator periods are uniformly distributed between $\theta{=}0$ (the no-action oscillator) and a max periodicity of $\theta{=}0.99 * T_C$.

In all T_C regimes, the accuracy improves as the oscillator density (number of oscillators per set respective to size of time interval) increases, which matches predictions based on biological precedence. This can be observed in Table III. The differing behavior in higher time resolutions is noteworthy, where accuracy increases with resolution alongside oscillator density. This may have something to do with a greater ability for oscillator periodicities to overlap and cover shorter periods. The weight distributes far more diffusely in the denser regimes, and with differing patterns of distribution across the different T_C. While the oscillator weights distribute with more variance and chaotic behavior over time, the solution stays stable and converges early.

VII. Conclusion

We have shown the SBF-A effectively addresses the periodicity finding problem in artificial systems. Our approach offers an efficient solution for identifying temporal patterns without requiring explicit temporal encoding or state comparisons. In our experiments, we demonstrate that the SBF-A successfully converges under multiple temporal regimes for the FI task, while also maintaining a high degree of computational efficiency and accuracy, supporting its potential applicability for temporally based RL tasks in dynamic real-world domains with extreme energy constraints. This opens the SBF-A framework to future exploration in more challenging and complex scenarios.

Metrics

Convergence: The ability to converge to a stable distribution of weights internally representing the solution.
Delay: Time to first correct action after T_C as a measure of accuracy. Lower is better.
Latency: Time to convergence. The earliest trial where peak Delay value is found, measured in time-steps. Lower is better.
Stability: The standard deviation in accuracy (Delay) after Convergence. Lower is better.
Energy: Total actions performed over the lifetime of the model, multiplied by the Latency. Lower is better.

References

[1] G. Buzsáki and M. Vöröslakos, "Brain rhythms have come of age," *Neuron*, vol. 111, no. 7, pp. 922–926, Apr. 5, 2023.
[2] F. Sawatani, K. Ide, and S. Takahashi, "The neural representation of time distributed across multiple brain regions differs between implicit and explicit time demands," *Neurobiology of Learning and Memory*, vol. 199, p. 107 731, Mar. 2023.
[3] J. E. Swearingen and C. V. Buhusi, "The Pattern of Responding in the Peak-Interval Procedure with Gaps: An Individual-Trials Analysis," *Journal of experimental psychology. Animal behavior processes*, vol. 36, no. 4, pp. 443–455, Oct. 2010.
[4] E. A. Petter, S. J. Gershman, and W. H. Meck, "Integrating Models of Interval Timing and Reinforcement Learning," *Trends in Cognitive Sciences*, Special Issue: Time in the Brain, vol. 22, no. 10, pp. 911–922, Oct. 1, 2018.
[5] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.
[6] L. Tallot and V. Doyère, "Neural encoding of time in the animal brain," *Neuroscience & Biobehavioral Reviews*, 2020.
[7] M. S. Matell and W. H. Meck, "Neuropsychological mechanisms of interval timing behavior," *BioEssays*, vol. 22, no. 1, pp. 94–103, 2000.
[8] M. S. Matell and W. H. Meck, "Cortico-striatal circuits and interval timing: Coincidence detection of oscillatory processes," *Brain Research. Cognitive Brain Research*, vol. 21, no. 2, pp. 139–170, Oct. 2004.
[9] C. V. Buhusi and W. H. Meck, "What makes us tick? Functional and neural mechanisms of interval timing," *Nature Reviews Neuroscience*, vol. 6, no. 10, pp. 755–765, 10 Oct. 2005.
[10] M. J. Allman and W. H. Meck, "Pathophysiological distortions in time perception and timed performance," *Brain*, vol. 135, pp. 656–677, Pt 3 Mar. 2012.
[11] B.-M. Gu, H. van Rijn, and W. H. Meck, "Oscillatory multiplexing of neural population codes for interval timing and working memory," *Neuroscience and Biobehavioral Reviews*, vol. 48, pp. 160–185, Jan. 2015.
[12] L. Zhang and P. Lin, "Reinforcement learning based energy-neutral operation for hybrid EH powered TBAN," *Future Generation Computer Systems*, vol. 140, pp. 311–320, Mar. 1, 2023.
[13] M. F. Reza, "Deep Reinforcement Learning Enabled Self-Configurable Networks-on-Chip for High-Performance and Energy-Efficient Computing Systems," *IEEE Access*, vol. 10, pp. 65 339–65 354, 2022.
[14] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 7540 Feb. 2015.
[15] N. Littlestone and M. K. Warmuth, "The Weighted Majority Algorithm," *Information and Computation*, vol. 108, no. 2, pp. 212–261, Feb. 1, 1994.
[16] W. Gerstner, M. Lehmann, V. Liakoni, D. Corneil, and J. Brea, "Eligibility Traces and Plasticity on Behavioral Time Scales: Experimental Support of NeoHebbian Three-Factor Learning Rules," *Frontiers in Neural Circuits*, vol. 12, 2018.
[17] T. Lattimore and C. Szepesvári, *Bandit algorithms*. Cambridge University Press, 2020.
[18] S. Han, M. Bendersky, P. Gajda, *et al.*, "Adversarial Bandits Policy for Crawling Commercial Web Content," in *Proceedings of The Web Conference 2020*, ser. WWW '20, New York, NY, USA: Association for Computing Machinery, Apr. 20, 2020, pp. 407–417.
[19] P. M. T. Broersen, *Automatic Autocorrelation and Spectral Analysis*. Springer Science & Business Media, Apr. 20, 2006, 301 pp.
[20] M. Vlachos, P. Yu, and V. Castelli, "On Periodicity Detection and Structural Periodic Similarity," in *Proceedings of the 2005 SIAM International Conference on Data Mining*, Philadelphia, PA: Society for Industrial and Applied Mathematics, Apr. 21, 2005.