Temporal Computing

Jonny Edwards

October 27, 2025

Abstract

This report consists of a review of the literature and a plan for integration related to **temporal computing**, a novel method of computation that uses time intervals as the primary method of storing and manipulating data. The literature review covers a broad range of methods, with the plan of further work focusing on assessing a novel temporally inspired digital implementation of a probabilistic temporal architecture, which can be readily used at the layer below the Metta language.

1 Introduction

The ability to perform accurate repetitive computation has been central to numerous scientific and technological advances in the last seventy years. At the heart of this is Moore's Law [44], which states that computational power provided by the silicon based Central Processing Units (CPUs) will double year-on-year for the foreseeable future. Unfortunately, several factors have compounded to make this less likely to continue: heat dissipation, atomic and quantum effects provide practical limits to the miniaturisation and packing of transistors [13], and the limited bandwidth between CPU and memory limits computation speed (the von-Neumann bottleneck [2]). There is also doubt about fabrication processes working below 3nm, although many vendors are issuing in even more miniaturisation which they refer to as the angstrom era (see Figure 1).

At the same time, methods of data processing using learning systems have become significant new architectural components. Of these, Deep Learning (DL) [32] has provided best-in-class performance on many machine learning tasks, and revitalised many areas of pattern recognition, opening up a revolution in new services that were traditionally performed using human intensive processes. Moreover, these methods are not well supported by traditional computer architectures [30] as they require massive parallelism, similarly to the brain. Graphics Processing Units (GPUs) , initially built for video games, have been successfully utilised, but still suffer from high levels of inefficiencies which are many orders of magnitude away from biological systems in energy and ability.

Given this impasse, there is a need to look more broadly at the problem of improving the efficiency **and** accelerating computation. This has resulted in a

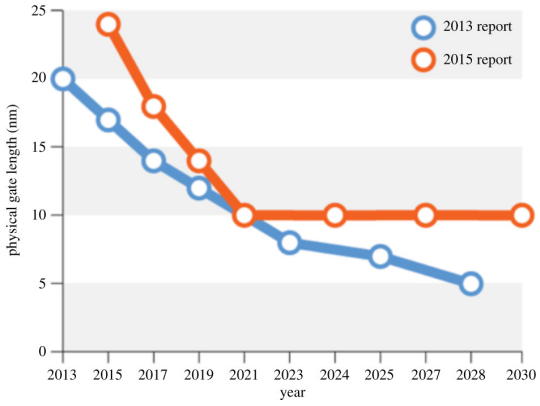

Figure 1: Predicted physical gate size decline over the next five years. Credit: `https://doi.org/10.1098/rsta.2019.0061/`

"Cambrian explosion" [28] of new approaches, and this research focuses on one of these, **temporal computing**.

This new approach represents numbers with a unary encoding (discussed in more detail in Section 3) instead of binary, and uses *time delays* to represent data instead of digital bit manipulation. We are drawn to this area for several reasons.

1. There is a precedence in nature, the human brain uses spike-based information as the main processing encoding [64].

2. Unary is more simplistic than binary to compute with. Tools such as the abacus are among the earliest calculators [47].

3. On the surface, time appears to be a free resource. For a typically coded temporal interval, the is only ever a requirement for two delimiting spikes (See Figure 2) [56].

The three capabilities detailed above provide a broad definition of an area of research, but this is yet to be established in the mainstream. Four areas: racelogic, unary processing, oscillatory Computing and neuromorphic compute, utilise time as a medium but do so without explicitly exploiting the above features. This is particularly the case with two areas under review, neuromorphic and oscillatory computing, with methods in these areas often conflating the benefits of hardware and problem specific algorithmic approaches.

The problem of defining temporal computing is further compounded by misuse of the term "stochastic" eg in works [67, 66], and significant oversight in not including previous work from the late sixties on unary compute [47], in discussions.

With this in mind, the agenda of this report is:

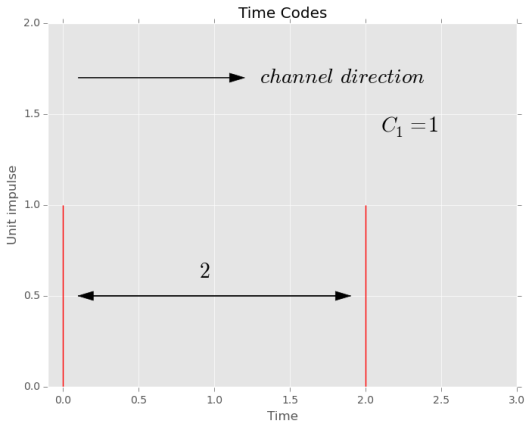

Figure 2: A typical value can be represented as two delimiting spikes.

- To cover all research specifically referenced as "temporal" by the racelogic community (section 2). For completeness, include Unary, oscillatory and neuromorphic computing in the discussion (Sections 6 and 7)
- To pave the way for the integration of this work into a broader probabilistic framework that is amenable for integration into a declarative AGI pipeline such as that provided by the Metta language.

2 Racelogic

Racelogic is undoubtedly the most significant area of research in the temporal domain, both computation and memory, and has formed a small community in the US [37].

The central theme of this work is the utilisation of the two timing properties of traditional logic gates. Under a temporal scheme an `AND` gate acts as a `MAX` operation, and the `OR` gate acts as a `MIN` operation (see Figure 3).

Using this a variety of systems have been proposed and implemented [37], most notably to perform minimum and maximum path search on a graph [38].

This work has been extended to utilise a tropical algebra [36] a semi-ring that implements multiplication and addition, and this was referred to as a Temporal State Machine. Under this scheme various additional graph algorithms including Dikstra's were implemented and benchmarked.

Further work by this group also involved building temporal memories called *Racetracks* [63, 39]. The basis of this is akin to dragging a bucket for a set distance, the time to do this becomes fixed for a set bucket size (see Figure 4). In this work this is implemented on a magnetic platform, with the addition of a recovery track to reset the memory. This is the first known implementation of memory and the "dragging time" approach seems to be something that could

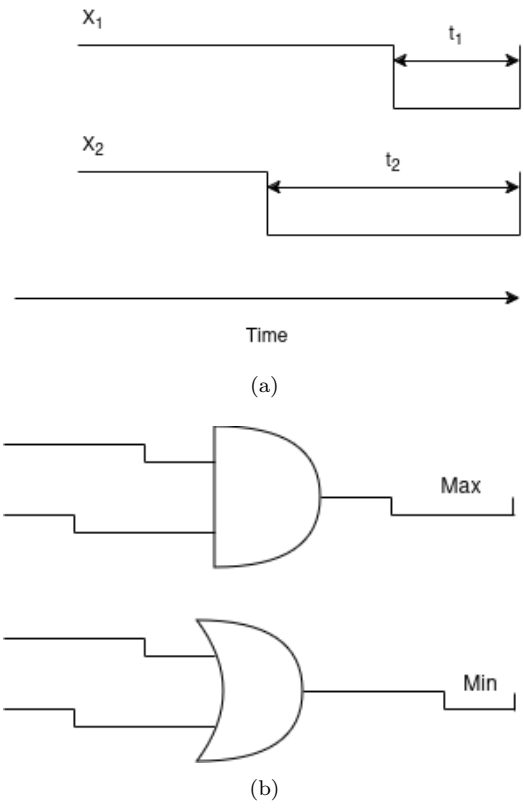

Figure 3: A visual explanation of racelogic. The first diagram shows the representation and second gives the "race" min/max effect

be readily applied in many resistive physical mediums.

The logic gate processing format has also been explored by two further parties, in the context of superconducting mediums [61] [62] [60] [65] [54] also as a formulation for a specific supporting algebra. The superconducting work follows a similar pattern and is limited to a constrained view of logic operations applied to temporally organised inputs. This work is excellent proof-of-concept work, but lacks reference to comparable digital algorithms and the nature of space/time trade-offs.

3 Unary Computation

Unary number systems ([34] Chapter 7 page 133) were amongst the earliest numerical representations of quantity, with the abacus existing as the earliest calculating device [47]. Unary codes are a simplification, with either binary digit acting as the delimiter. Table 1 gives a breakdown of how unary codes

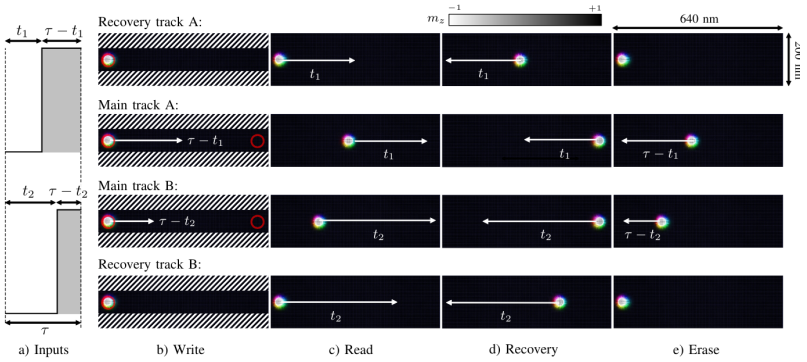

Figure 4: The racetrack memory operations. Taken from [63]

are constructed for the first few decimal numbers. We introduce a further code Discrete Pulse Interval Modulation (DPIM), which is a form of unary code, and is used in the communication community.

Decimal	Unary (U)	Unary code (UC)	DPIM
0		(0) may delimit	(11) may delimit
1	1	1	101
2	11	10	1001
3	111	100	10001

Table 1: Coding Schemes related to Temporal Computing

One of the key benefits for all the unary family is that simple arithmetic is easily derived: Addition with unary, as with the abacus, is performed trivially using simple concatenation [18].

Integer multiplication of unary codes is performed by dilating the first unary code by the size of the second code. In a fully clock based architecture this can be performed by changing the relative clock speeds of the sender and receiver of the signal (see [18] section 3.3 for a more detailed explanation) Again, this is a simple operation which requires only basic logic to perform.

The review paper of Poppelbaum [47] explains in some detail first-generation unary computers. There are three types of unary processing: stochastic, burst and pulse gear based approaches. These systems can be summarised as follows:

Stochastic where the unary value represents a discrete probabilistic range, sometimes with randomness incorporated. Many machines have been built with this premise, including **RASCEL** [20] **POSTCOMP** [46]. This continues to be an active field of research [1]. We review this work in more detail below.

Burst This is more akin to traditional unary but with a specific bit-width, so for instance 7 out of 10 bits set (`1111111000`) would equal 7. Examples of it's use are **BURST** [47] and **WALSHSTORE** [4].

Pulse Gear This again uses burst unary but provides a clever method for shifting and ANDing to perform operations. An example of this is the **UNIFIELD** processor [17].

These largely forgotten systems are all very interesting applications of unary and could be utilised in the design of new processing elements, the Pulse Gear methods, and the use of AND gates and counters have some resonance with methods discussed later in this review.

Two further points should be noted. Firstly, none of these systems use the benefits of redundancy and the delimiting nature of unary codes and are firmly wedded to using many boolean `on` (`1`'s) values. Secondly they are explicitly non-temporal, this means that time is not used to represent data.

More modern work in this area has been performed by Wu [67] [45] [68]. This work is again mainly unary and again not explicitly temporal.

Before moving on to more elaborate unary notations, let us briefly discuss stochastic and probabilistic methods. In stochastic computing,a form of unary encoding is performed using random bitstreams where the proportion of 1s represents a numerical value. This is typically done using either **unipolar** or **bipolar** encoding schemes. Unipolar Encoding represents values in the range $[0, 1]$. A number x is encoded as a bitstream where the probability of a 1 is x. For example, a value of 0.75 might be encoded as a bitstream like `1101111011111011`, which contains approximately 75% 1s. A Bipolar Encoding Represents values in the range $[-1, +1]$. A value x is encoded with the probability of a 1 as $(x + 1)/2$. For example, a value of -0.5 would be encoded with a bitstream containing approximately 25% 1s.

3.0.1 Advantages

- **Simplified arithmetic:** Multiplication becomes a simple AND gate (unipolar) or XNOR gate (bipolar).
- **Fault tolerance:** Randomness makes systems robust to noise and bit flips.
- **Low hardware cost:** Uses simple logic gates instead of full arithmetic units.

4 Example: Unipolar Multiplication

To multiply 0.5×0.75:

- Encode 0.5 as a bitstream with 50% 1s.
- Encode 0.75 as a bitstream with 75% 1s.

- AND the streams to get a result with approximately 37.5% 1s, representing 0.375.

5 Challenges

- **Accuracy:** Requires long bitstreams for precision.
- **Latency:** Longer streams mean slower computation.
- **Correlation:** Input streams must be uncorrelated to avoid bias.

A **p-bit** (probabilistic bit) is a classical binary unit that fluctuates randomly between $+1$ and -1 with a tunable probability. Unlike deterministic bits, p-bits are inherently stochastic and can be biased toward one state or the other using an input signal [7].

5.0.1 Mathematical Model

The state m_i of a p-bit is given by:

$$m_i(t) = +/- [\tanh(I_i(t)) + r(t)] \tag{1}$$

where:

- $I_i(t)$ is the input bias at time t
- $r(t)$ is a random number uniformly distributed in $[-1, 1]$
- $+/-$ is the sign function

5.0.2 Key Properties

- **Tunable randomness:** The probability of outputting $+1$ or -1 depends on the input bias.
- **Autonomous flipping:** P-bits update asynchronously and independently [58].
- **Energy-efficient:** Can be implemented using low-power devices like stochastic magnetic tunnel junctions (sMTJs) [8].

5.0.3 Comparison to Stochastic Computing

While both p-bits and stochastic computing rely on randomness, p-bits Operate on single fluctuating bits rather than long bitstreams, and allow dynamic tuning of probability via input bias. They are better suited for asynchronous, parallel architectures.

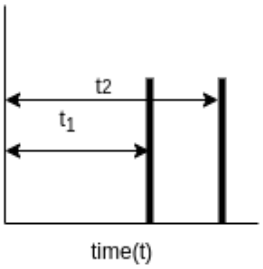

Figure 5: An example of genuine multiplexing

5.1 Multiplexed Unary

Unary codes have an additional property that a set of non-duplicate values can be efficiently multiplexed. Again to use the abacus analogy, if we have beads of two colours: black (1) and white (0), provided there is no duplication, multiplexing can occur which *reuses white beads*. Again, the logic of this is a simple `OR`ing of the value from a start reference bit. Figure 5 presents this graphically. This system is *genuinely* multiplexing; the two values are overlaid so the values actually forms part of the subsequent data. This is unique to this coding strategy and is unlike any other system of multiplexing. The general algorithm for conversion to this multiplexed domain is as follows:

```
1 for each value I in C do
2 |  At position I in MUX window append 1;
3 end
```

Algorithm 1: Encoding Algorithm

Once transformed, signals can be interleaved in such a way that they can be easily recovered. To de-multiplex, the process is reversed, with the MUX channel progressively reconstructs the input values back into separate channels.

```
1 for 0..I do
2 |  if 1 then
3 |  |  Write I into output channel;
4 |  end
5 end
```

Algorithm 2: Decoding Algorithm for DMU

A natural extension is to enable the unary coded 1 value to be multi-valent. Although at first this may appear contrived, the motivation for this is the extension of the multiplexing method to incorporate duplicates, and the ability to better encode analog interval channels. This channel also has an additional

property that multiplication can be performed, at least within the bounds of the notation, by placing a value a at position t, and *also* multiplexing as in Figure 6.

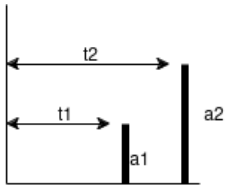

Figure 6: Multi-valent multiplexing for a potential analog implementation

Although not immediately obvious, an uncertainty relationship exists between position and data when "dense" multiplexing is used. To illustrate this let us take the trivial sequence (1,3,2) with knowledge of the prior this could be easily transformed into the densely multiplexed representation of (111) [1] which notionally can be represented with a smaller number of bits, but with a loss of order in the data. Adding positional values via a positional index recovers from the problem but at the cost of extra data, with a different prior distribution.

Clearly, the act of multiplication can be transformed into manipulating the values held in the buckets. This can be easily extended to a dot-product operation and the representation of tuple pairs is somewhat compressive since one element of the 2-tuple is specified by a positional index.

Under a simple scheme, tuples can be added arbitrarily to the array at their t index point. A dot product on this multiplexed array can be evaluated efficiently and in a systolic way using the **Multiplicative ADD** (MADD) algorithm (Algorithm 3) as shown in [19]. Figure 7 demonstrates that the algorithm is effectively finding the area of the rectangles 4×10 and 4×4 by progressively sweeping in one pass back across the indexed array, counting initially the first 6×4 and then the remaining 8×4.

```
1 acc, height ← 0;
2 i ← |mem| − 1;
3 while i > 0 do
4 |   height ← height + mem[i];
5 |   acc ← acc + height;
6 |   i ← i − 1;
7 end
8 return acc
```

Algorithm 3: The MADD algorithm

The MADD algorithm originated from the use of discrete memory indexes as a simulation of time delays. As such the MADD algorithm is inspired by the

[1] 1=1,2=01,3=001

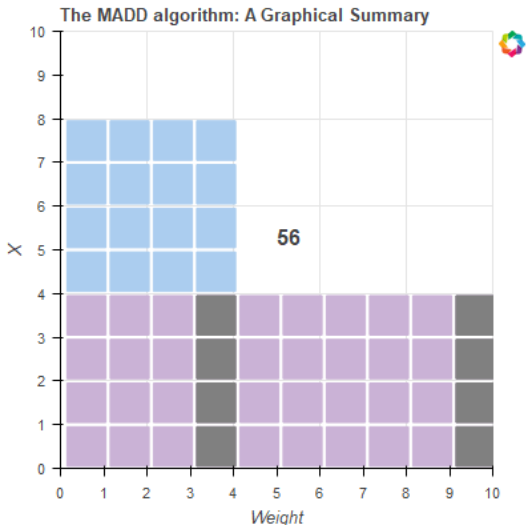

Figure 7: MADD: A description based on a geometric view of the MADD algorithm. Here we perform the operation $(4 \times 10) + (4 \times 4)$ right to left by accumulating the "height" at the index point

temporal approach, but, its representation does not use time delays directly as the storage medium.

6 Oscillatory Systems

An oscillatory computer utilises the dynamics of time-based coupled oscillators to perform computation.

Coupled Oscillators are a technique in non-linear dynamics that produce stable and predictable patterns of oscillation from the ensemble behaviours of many oscillators [57]. The classic example of this is the firefly synchronisation problem [5] where fireflies in a forest synchronise their flashing patterns.

Unlike traditional computers, oscillatory computers encode and process information using the phase, frequency, or amplitude of oscillations. This approach is inspired by the observation that many biological systems, including the brain, use oscillatory dynamics for information processing and communication. For a review article and example see [11] and [10]. Example systems include [41] [15] [16] [9] [22] [40].

The advantages of oscillatory computers include:

- Parallelism through the emerging dynamics of coupling.
- Energy Efficiency as they often rely on the natural dynamics of physical systems.

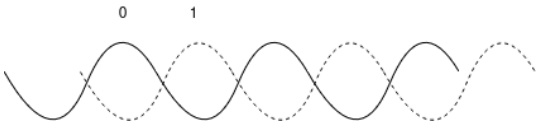

Figure 8: An example of how phase might encode a signal, in this case binary 0 and 1

- Robustness to noise and variability, due to the collective dynamics of coupled oscillators.

Some of challenges associated with oscillatory computers include:

Precision The precision of computations in oscillatory computers may be limited by the inherent variability and noise in the oscillatory dynamics.

Programmability Programming oscillatory computers can be difficult, as it requires the control of the complex dynamics of the coupling.

Scalability Scaling up oscillatory computers to large numbers of oscillators can be difficult, as the dynamics can become increasingly complex and hard to control.

Interfacing as with many systems discussed in this document interfacing with traditional digital systems is problematic, with similar problems to analog to digital conversion.

7 Spiking Neural Networks and Neuromorphic Compute

Spiking Neural Networks (SNNs) [33] are a type of artificial neural network that more closely mimic the behavior of biological neural networks. SNNs are a key component of neuromorphic computing, which aims to develop hardware and software systems that mimic the structure and function of biological neural networks. This is extensive literature on neuromorphic hardware platforms, such as SpiNNaker [23, 24], TrueNorth [52], Loihi [14], and BrainScale [53]. These platforms are optimized for low-power, real-time operation often outperforming their DL cousins in terms of energy usage [48], but at the cost of conjoined hardware and software offerings.

SNNs are explicitly temporal using spikes to define data, their key aspects are:

Information Encoding There are a variety of coding schemes, not just the "width" encoding discussed in this document. These are:

Rate Coding Information is encoded in the average firing rate of neurons, similar to traditional ANNs.

Temporal Coding Information is encoded in the precise timing of individual spikes. This can include schemes like Time to First Spike (TTFS) or Rank Order Coding (ROC).

Population Coding Information is encoded in the collective activity of a group of neurons.

Neuron Models SNNs use more biologically plausible activation functions than ANNs. Some commonly used ones are:

Integrate-and-Fire (IF) Neurons [6] IF neurons integrate incoming spikes and fire an output spike when a threshold is reached. After firing, the neuron's membrane potential is reset.

Leaky Integrate-and-Fire (LIF) Neurons [59] LIF neurons are similar to IF neurons but include a leakage term that causes the membrane potential to decay over time.

Hodgkin-Huxley Neurons [69] The Hodgkin-Huxley model is a detailed biophysical model that describes the dynamics of ion channels in the neuron membrane.

Synaptic Models In SNNs, synapses can have dynamic behaviors, unlike the static weights used in traditional ANNs. This gives rise to a computationally richer and more compact pattern processing systems.

One of the most relevant applications of this idea is the work done by Hopfield [29]. Hopfield's key insight was that the correct representation is the key to solving difficult problems:

> The choice of information representation is of vital importance in making a computation easy, as is illustrated by the question 'is 3630225 divisible by 7?'. If the number 3630225 is in base 10, answering this takes a bit of work. If the number 3630225 is in base 7, the answer is immediate from inspection of the last digit. The choice of representation for analogue information is equally important for neural computation. [29]

The main focus of this work is to mimic the behaviour of the olfactory system. This particular problem is termed the "analog match" problems, and whilst solvable via traditional logistic methods is more robust when recast as a coincidence problem. Figure 9 shows how the system functions, note that the data is log scaled.

8 MeTTa: A Reflective Meta-Language for AGI

MeTTa (Meta-Type Talk) [27] [55] is the foundational programming language of OpenCog Hyperon [21], designed to support Artificial General Intelligence (AGI) through symbolic reasoning, introspection, and self-modifying code. It serves as a meta-language capable of defining and manipulating other languages and type systems.

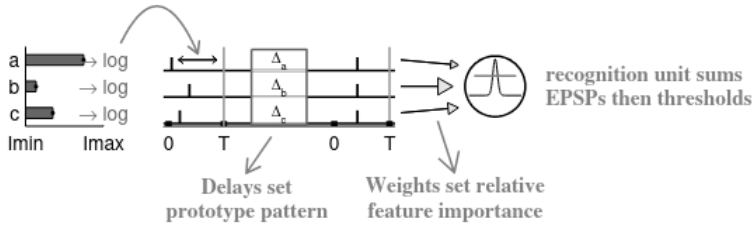

Figure 9: Computing with Action Potentials, taken from [29]

8.1 Core Features

Reflective Metagraph Rewriting, Programs are represented as metagraphs—graphs of graphs—enabling recursive transformations and dynamic reasoning. MeTTa uses powerful pattern-matching and substitution primitives to infer and rewrite expressions. The language is syntactically sparse, allowing users to define custom logics and type systems. Grounding Support: Symbolic atoms can be linked to sensory or motor data, enabling embodied cognition. MeTTa programs can inspect and alter their own structure, supporting recursive self-improvement.

MeTTa operates as the lowest-level cognitive substrate in OpenCog Hyperon. It interfaces with the Distributed Atomspace (DAS), orchestrating reasoning, learning, and control. It replaces Atomese from OpenCog Classic with a more flexible and semantically rich alternative.

Applications

- Symbolic reasoning and logic programming
- Cognitive architecture modeling
- AGI-oriented introspection and learning

MeTTa, the meta-language of OpenCog Hyperon, supports probabilistic reasoning by enabling symbolic manipulation of uncertain knowledge through pattern rewriting and weighted truth values. Unlike traditional probabilistic programming languages that rely on explicit probability distributions, MeTTa operates over a hypergraph-based memory structure (the Distributed Atomspace), where each atom may carry a *truth value*—a pair representing strength and confidence.

Probabilistic inference in MeTTa emerges from the interaction of three key mechanisms: (1) pattern matching over uncertain structures, (2) rule-based rewriting conditioned on probabilistic thresholds, and (3) recursive self-modification of inference strategies. These mechanisms allow MeTTa to encode Bayesian-like updates, fuzzy logic, and stochastic search within a unified symbolic substrate.

Rather than treating probability as a separate computational layer, MeTTa integrates it into the fabric of cognitive processes. For example, attention allocation, memory retrieval, and goal selection can all be guided by probabilistic heuristics encoded as MeTTa rules. This enables AGI systems to reason under uncertainty, adaptively revise beliefs, and simulate human-like cognition.

9 Probabilistic Reasoning Example in MeTTa

10 Problem

Decide whether to carry an umbrella based on the probability of rain. If the probability exceeds 0.6, the system recommends carrying an umbrella.

11 MeTTa-Like Representation

```
;; Define a probabilistic belief
(Probability (RainToday) 0.75)

;; Rule: If rain probability exceeds threshold, recommend umbrella
(If
  (GreaterThan (Probability (RainToday)) 0.6)
  (Then (Action CarryUmbrella))
)
```

12 Rewrite Logic

```
;; Pattern to match
(Probability ?event ?p)
(GreaterThan ?p 0.6)

;; Rewrite result
(Action CarryUmbrella)
```

13 Truth Value Representation

In OpenCog Hyperon, atoms may carry truth values as pairs of strength and confidence:

```
(RainToday)
:TruthValue (Strength 0.75) (Confidence 0.9)
```

14 Interpretation

This structure allows MeTTa to reason symbolically over uncertain inputs and take action based on probabilistic thresholds. The system can generalize this logic to other decisions involving uncertainty.

15 Critique

The document provides an overview of four emerging areas in computing: race logic, unary, oscillator, and neuromorphic. Each area presents unique challenges and opportunities for advancement in computational technology. To summarise the key points:

Racelogic is a practical and promising approach for utilising temporal data in computing. It's highlighted as one of the first methods to effectively a *truly* temporal approach. Also, the work on temporal memories, significantly strengthens the case for this technology, and racelogic has already explored wide-ranging applications. Its scope in terms of arithmetic and compute based operations is currently limited to a few comparison operations, and hence there is a lack of insight into utilising race logic as a general-purpose computer.

Oscillatory Computing Oscillator computing shows significant promise, with potential for high-speed operations in the range of 50-100 GHz. This technology could potentially offer a 10x improvement over current computing capabilities, however seems to be applicable to specialist problems rather than general compute.

Neuromorphic Computing is the most mature technology among those reviewed. However, it is perhaps too literal in mimicking brain functionality, which may limit its potential. Our view is that time-based approaches should be emphasised over biological plausibility the analogy we draw is that of aircraft design, which draws inspiration from birds flight but in no way mimics the wing-flapping approach.

Unary Computing Despite extensive past research, unary computing has been largely overlooked in recent years. It holds potential for reevaluation in light of advancements in AI and complex computing systems. The primary challenge lies in its memory footprint, as traditional memory doesn't scale well for unary representations. However, true temporal systems, ones which use time as data directly, are an avenue to alleviate this problem. Future research opportunities also include exploring unary signals with non-linear representations, which increase the efficiency of the subsequent computation. It is this methodology together with the discussion on pt-systems that provides the most synergy with the Metta language, and therefore the primary recommendation is to exploit this more fully.

In conclusion, while each area reviewed above faces unique challenges, the pt-systems work is the most salient for further investigation, and where the scope should be more narrowly focused. Our hope is that a fully temporal stochastic system will allow us position Metta as the top level language for probabilistic stochastic computing.

To finalise next steps, providing this report is accepted, investigations will start into how to implement a pt-system as low level hardware, using existing ideas discussed in this report but alleviating the memory constraints by building a truly temporal system.

References

[1] A. Alaghi and J. P. Hayes. Survey of stochastic computing. *ACM Transactions on Embedded computing systems (TECS)*, 12(2s):1–19, 2013.

[2] J. Backus. Can programming be liberated from the von neumann style?: A functional style and its algebra of programs. *Commun. ACM*, 21(8):613–641, Aug. 1978.

[3] A. Borst and F. E. Theunissen. Information theory and neural coding. *Nature neuroscience*, 2(11):947–957, 1999.

[4] E. Bracha. *Walshstore: the application of burst processing to fail-soft storage systems using walsh transforms.* University of Illinois at Urbana-Champaign, 1978.

[5] J. Buck. Synchronous rhythmic flashing of fireflies. ii. *The Quarterly review of biology*, 63(3):265–289, 1988.

[6] A. N. Burkitt. A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biological cybernetics*, 95:1–19, 2006.

[7] K. Y. Camsari, S. Salahuddin, and S. Datta. Stochastic p-bits for invertible logic. *Physical Review X*, 7(3):031014, 2017.

[8] S. Chowdhury, K. Y. Camsari, and S. Datta. A full-stack view of probabilistic computing with p-bits. *arXiv preprint arXiv:2301.06606*, 2023.

[9] E. Corti. Networks of coupled vo2 oscillators for neuromorphic computing. Technical report, EPFL, 2021.

[10] G. Csaba and W. Porod. Perspectives of using oscillators for computing and signal processing. *arXiv preprint arXiv:1805.09056*, 2018.

[11] G. Csaba and W. Porod. Coupled oscillators for computing: A review and perspective. *Applied physics reviews*, 7(1), 2020.

[12] J. Das and P. D. Sharma. Pulse-interval modulation. *Electronics Letters*, 3:288–289, 1967.

[13] R. David. We're not prepared for the end of moore's law. https://www.technologyreview.com/2020/02/24/905789/were-not-prepared-for-the-end-of-moores-law/. Accessed: 1 Jan 2024.

[14] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[15] C. Delacour, S. Carapezzi, M. Abernot, G. Boschetto, N. Azemard, J. Salles, T. Gil, and A. Todri-Sanial. Oscillatory neural networks for edge ai computing. In *2021 IEEE computer society annual symposium on VLSI (ISVLSI)*, pages 326–331. IEEE, 2021.

[16] C. Delacour, S. Carapezzi, G. Boschetto, M. Abernot, T. Gil, N. Azemard, and A. Todri-Sanial. A mixed-signal oscillatory neural network for scalable analog computations in phase domain. *Neuromorphic Computing and Engineering*, 3(3):034004, 2023.

[17] A. Dollas. *Architecture and applications of a unifield-type computer*. University of Illinois at Urbana-Champaign, 1987.

[18] J. Edwards, S. O'Keefe, and W. D. Henderson. Unconventional arithmetic: A system for computation using action potentials. In *Proc. of Unconventional Computation and Natural Computation*, pages 155–163, 2014.

[19] J. Edwards, A. Wheeldon, R. Shafik, and A. Yakovlev. A variable bitwidth asynchronous dot product unit. In *IEEE International Symposium on Asynchronous Circuits and Systems*, 2019.

[20] J. W. Esch. *—Rascel—A Programmable Analog Computer Based on a Regular Array of Stochastic Computing Element Logic*. University of Illinois at Urbana-Champaign, 1969.

[21] O. Foundation. Opencog hyperon documentation. https://opencog.org/hyperon, 2025. Project documentation.

[22] E. P. Frady and F. T. Sommer. Robust computation with rhythmic spike patterns. *Proceedings of the National Academy of Sciences*, 116(36):18050–18059, 2019.

[23] S. Furber. Large-scale neuromorphic computing systems. *Journal of Neural Engineering*, 13(5):051001, 2016.

[24] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana. Spinnaker: A million core processor system for highly parallel neural network simulation. *Proceedings of the IEEE*, 102(5):652–665, 2014.

[25] Z. Ghassemlooy and A. Hayes. Digital pulse interval modulation for ir communication systems—a review. *International Journal of Communication Systems*, 13(7-8):519–536, 2000.

[26] Z. Ghassemlooy, A. Hayes, N. Seed, and E. Kaluarachchi. Digital pulse interval modulation for optical communications. *IEEE Communications Magazine*, 36(12):95–99, 1998.

[27] B. Goertzel. Reflective metagraph rewriting as a foundation for an agi language of thought. *OpenCog Foundation*, 2021. White paper.

[28] J. L. Hennessy and D. A. Patterson. A new golden age for computer architecture. *Communications of the ACM*, 62(2):48–60, 2019.

[29] J. J. Hopfield. Pattern recognition computation using action potential timing for stimulus representation. *Nature*, 376(6535):33–36, 1995.

[30] W. Jeon, G. Ko, J. Lee, H. Lee, D. Ha, and W. W. Ro. Chapter six - deep learning with gpus. In S. Kim and G. C. Deka, editors, *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning*, volume 122 of *Advances in Computers*, pages 167–215. Elsevier, 2021.

[31] E. D. Kaluarachchi. *Digital pulse interval modulation for optical communication systems*. Sheffield Hallam University (United Kingdom), 1997.

[32] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[33] W. Maass and C. M. Bishop. *Pulsed neural networks*. MIT press, 2001.

[34] D. J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

[35] D. M. MacKay and W. S. McCulloch. The limiting information capacity of a neuronal link. *The bulletin of mathematical biophysics*, 14(2):127–135, 1952.

[36] A. Madhavan, M. W. Daniels, and M. D. Stiles. Temporal state machines: Using temporal memory to stitch time-based graph computations. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 17(3):1–27, 2021.

[37] A. Madhavan, T. Sherwood, and D. Strukov. Race logic: A hardware acceleration for dynamic programming algorithms. *ACM SIGARCH Computer Architecture News*, 42(3):517–528, 2014.

[38] A. Madhavan, T. Sherwood, and D. Strukov. Race logic: abusing hardware race conditions to perform useful computation. *IEEE Micro*, 35(3):48–57, 2015.

[39] A. Madhavan and M. D. Stiles. Storing and retrieving wavefronts with resistive temporal memory. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020.

[40] O. Maher, R. Bernini, N. Harnack, B. Gotsmann, M. Sousa, V. Bragaglia, and S. Karg. Highly reproducible and cmos-compatible vo2-based oscillators for brain-inspired computing. *Scientific Reports*, 14(1):11600, 2024.

[41] M. Mandal and B. C. Sarkar. Ring oscillators: Characteristics and applications. *Indian Journal of Pure and Applied Physics*, 48:136–145, 02 2010.

[42] S. Marougi and K. Sayhood. Signal-to-noise performance of the pulse-interval and width-modulation system. *Electronics Letters*, 14(19):528–530, 1983.

[43] M. Meister and M. J. Berry. The neural code of the retina. *Neuron*, 22(3):435–450, 1999.

[44] G. E. Moore. Readings in computer architecture. chapter Cramming More Components Onto Integrated Circuits, pages 56–59. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[45] Z. Pan and J. S. M. Di Wu. T-mac: Temporal multiplication with accumulation. *TC*, 1:2, 2022.

[46] W. Poppelbaum, C. Afuso, and J. Esch. Stochastic computing elements and systems. In *Proceedings of the November 14-16, 1967, fall joint computer conference*, pages 635–644, 1967.

[47] W. Poppelbaum, A. Dollas, J. Glickman, and C. O'Toole. Unary processing. In M. C. Yovits, editor, *Advances in Computers*, volume 26 of *Advances in Computers*, pages 47 – 92. Elsevier, 1987.

[48] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner. Survey of machine learning accelerators. 09 2020.

[49] F. Rieke, D. Bodnar, and W. Bialek. Naturalistic stimuli increase the rate and efficiency of information transmission by primary auditory afferents. *Proceedings of the Royal Society of London B: Biological Sciences*, 262(1365):259–265, 1995.

[50] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[51] M. Sato, M. Murata, and T. Namekawa. Pulse interval and width modulation for video transmission. *IEEE Transactions on Cable Television*, (4):165–173, 1978.

[52] J. Sawada, F. Akopyan, A. S. Cassidy, B. Taba, M. V. Debole, P. Datta, R. Alvarez-Icaza, A. Amir, J. V. Arthur, A. Andreopoulos, et al. Truenorth ecosystem for brain-inspired computing: scalable systems, software, and applications. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 130–141. IEEE, 2016.

[53] S. Schmitt, J. Klähn, G. Bellec, A. Grübl, M. Guettler, A. Hartel, S. Hartmann, D. Husmann, K. Husmann, S. Jeltsch, et al. Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system. In *2017 international joint conference on neural networks (IJCNN)*, pages 2227–2234. IEEE, 2017.

[54] C. H. Segal. *Digital Non-binary Spiking Communication and Computation Channel*. University of California, Santa Barbara, 2022.

[55] SingularityNET. Metta in a nutshell. `https://opencog.org/hyperon`, 2024. Overview of MeTTa language for OpenCog Hyperon.

[56] J. E. Smith. Temporal computer organization. *arXiv preprint arXiv:2201.07742*, 2022.

[57] S. H. Strogatz. Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering (studies in nonlinearity). *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering (Studies in Nonlinearity)*, 2001.

[58] B. Sutton, R. Faria, L. A. Ghantasala, R. Jaiswal, K. Y. Camsari, and S. Datta. Autonomous probabilistic coprocessing with petaflips per second. *arXiv preprint arXiv:1907.09664*, 2019.

[59] D. Tal and E. L. Schwartz. Computing with the leaky integrate-and-fire neuron: logarithmic computation and multiplication. *Neural computation*, 9(2):305–318, 1997.

[60] G. Tzimpragos. *Computing with Temporal Operators*. University of California, Santa Barbara, 2022.

[61] G. Tzimpragos, N. Tsiskaridze, K. Huch, A. Madhavan, and T. Sherwood. From arbitrary functions to space-time implementations. In *Proceedings of the 1st Unary Computing Workshop (ISCA'19)*, 2019.

[62] G. Tzimpragos, J. Volk, D. Vasudevan, N. Tsiskaridze, G. Michelogiannakis, A. Madhavan, J. Shalf, and T. Sherwood. Temporal computing with superconductors. *IEEE Micro*, 41(3):71–79, 2021.

[63] H. Vakili, M. N. Sakib, S. Ganguly, M. Stan, M. W. Daniels, A. Madhavan, M. D. Stiles, and A. W. Ghosh. Temporal memory with magnetic racetracks. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 6(2):107–115, 2020.

[64] J. D. Victor. How the brain uses time to represent and process visual information. *Brain research*, 886(1-2):33–46, 2000.

[65] J. Volk, A. Wynn, E. Golden, T. Sherwood, and G. Tzimpragos. Addressable superconductor integrated circuit memory from delay lines. *Scientific Reports*, 13(1):16639, 2023.

[66] D. Wu. *Power-Efficient Computer Architecture via Unary and Approximate Computing*. The University of Wisconsin-Madison, 2023.

[67] D. Wu, J. Li, R. Yin, H. Hsiao, Y. Kim, and J. San Miguel. Ugemm: Unary computing architecture for gemm applications. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 377–390. IEEE, 2020.

[68] D. Wu and J. San Miguel. usystolic: Byte-crawling unary systolic array. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 12–24. IEEE, 2022.

[69] B. A. y Arcas, A. L. Fairhall, and W. Bialek. Computation in a single neuron: Hodgkin and huxley revisited. *Neural computation*, 15(8):1715–1749, 2003.