Journal of Physics:
Complexity

What does it mean for a system to compute?

To cite this article: David H Wolpert and Jan Korbel 2026 *J. Phys. Complex.* **7** 015010

IOP Publishing J. Phys. Complex. 7 (2026) 015010 https://doi.org/10.1088/2632-072X/ae3af8

Journal of Physics: Complexity

PAPER

What does it mean for a system to compute?

David H Wolpert[1,2,3,4,5] and Jan Korbel[2,*]

[1] Santa Fe Institute, Santa Fe, NM 87501, United States of America
[2] Complexity Science Hub, Metternichgasse 8, 1030 Vienna, Austria
[3] ICTP—The Abdus Salam International Centre for Theoretical Physics, Strada Costiera 11, 34151 Trieste, Italy
[4] Albert Einstein Institute for Advanced Study, New York, NY, United States of America
[5] Current address: Arizona State University, Tempe AZ, 85287, United States of America
[*] Author to whom any correspondence should be addressed.

E-mail: korbel@csh.ac.at and david.h.wolpert@gmail.com

Keywords: dynamical systems, computation, non-constructed computers

OPEN ACCESS

RECEIVED
19 September 2025

REVISED
8 January 2026

ACCEPTED FOR PUBLICATION
20 January 2026

PUBLISHED
18 February 2026

Abstract

Many real-world dynamic systems, both natural and artificial, are understood to be performing computations. For artificial dynamic systems, explicitly designed to perform computation—such as digital computers—by construction, we can identify which aspects of the dynamic system match the input and output of the computation that it performs, as well as the aspects of the dynamic system that match the intermediate logical variables of that computation. In contrast, in many naturally occurring dynamic systems that we understand to be computers, even though we neither designed nor constructed them—such as the human brain—it is not *a priori* clear how to identify the computation we presume to be encoded in the dynamic system. Regardless of their origin, dynamic systems capable of computation can, in principle, be mapped onto corresponding abstract computational machines that perform the same operations. In this paper, we begin by surveying a wide range of dynamic systems whose computational properties have been studied. We then introduce a very broadly applicable framework for identifying what computation(s) are emulated by a given dynamic system. After an introduction, we summarize key examples of dynamic systems whose computational properties have been studied. We then introduce a very broadly applicable framework that defines the computation performed by a given dynamic system in terms of maps between that system's evolution and the evolution of an abstract computational machine. We illustrate this framework with several examples from the literature, in particular discussing why some of those examples do *not* fully fall within the remit of our framework. We also briefly discuss several related issues, such as uncomputability in dynamic systems, and how to quantify the 'value of computation' in naturally occurring computers. We conclude with a discussion of some of the promising directions for future research.

1. Introduction

1.1. Background

From neural networks in simple organisms to modern cloud computer systems to human social systems, computational processes are pervasive in both natural and artificial dynamic systems [1–4]. Centuries of research have significantly advanced our theoretical understanding of some of these computational processes. In particular, mathematical developments have deepened our understanding of the computability [5], computational complexity [6], and (resource-bounded) Kolmogorov complexity [6, 7] of computational tasks. These developments are exemplified by deep issues like the famous **P** versus **NP** problem.

In addition, there has been great progress in our ability to design and then physically construct dynamic systems to implement specific computations. To do this, we identify the initial (perhaps coarse-grained) state of the dynamic system with the desired input to the algorithm, and its final state (if it exists) is identified with the output of the computation. So changing the actual dynamics of the system constitutes changing the computation itself.

Crucially, in such human-constructed computers, we choose the **decoding** map, taking the degrees of freedom of the dynamic system to the logical variables in the abstract computer we wish to view that system as implementing. This means that the relationship between such a system's dynamics and the computer it is implementing is explicitly known before the dynamic system starts its evolution. In short, we have an *a priori computational blueprint* for mapping the dynamics of the dynamic system to that of a computer.

As a point of contrast, consider naturally occurring dynamic systems that are often viewed as performing computations, even though humans did not explicitly construct those systems to perform some specific computation. Examples of such dynamic systems range from individual cells to entire brains to turbulent flows to whole human societies. Even systems like off-equilibrium spin networks have been viewed as systems performing computations [8].

In these naturally occurring systems, there is no pre-specified decoding map taking the degrees of freedom of the dynamic system into the logical variables of an abstract computational machine. Moreover, the same dynamic system can often be viewed as performing different computations, depending on how we choose to map the dynamic system's variables to logical variables in a computational machine, i.e. how to decode those variables. (This is true of both human-constructed dynamic systems and naturally occurring ones.) More precisely, changing that map not only changes how we interpret the dynamic system's initial state as the input to a computer; it can also change how we interpret the subsequent dynamics of that system as a computation unfolding.

In fact, many (fixed) dynamic systems can be viewed as performing an infinite set of different computations, depending on our choice of the decoding map. A proof sketch of this is given in section 4.1 (see also [9]).

1.2. Constructed versus non-constructed computers

Our goal with this paper is to formalize exactly how one can decode the states of a dynamic system into the logical variables in a computer, and then to review examples of such dynamic systems.

To begin, we highlight the fact that intuitively speaking, there are two different types of dynamic systems we can interpret as performing a computation. We will not formally define a distinction between these two types of computers. Rather, we introduce this distinction because it is useful for thinking about the issue of how to identify the computation(s) implemented by any given dynamic sense.

In the first type of dynamic system, the decoding of some of the dynamic system's variables as the logical variables of a computational machine is known ahead of time, before that dynamic system actually undergoes its dynamics. In this paper, we are interested in such dynamic systems that are physical devices arising in the real world, where the decoding function is chosen by a human engineer or scientist who has designed and then constructed that physical device, with the explicit goal of using the device together with that decoding function to implement some specific computational machine. We will refer to dynamic systems of this type as **constructed computers**. Ultimate examples of constructed computers are the digital devices that underlie modern society. See, e.g. [10] for a recent discussion.

The second type of dynamic system does not involve a human designing and then constructing a physical device with an explicit goal of having it implement a desired computation. Rather, it is the goal of the human investigating such systems to infer a decoding of some of the variables of that dynamic system as the logical variables of a computational machine. We will refer to dynamic systems of this second type as **non-constructed computers**. For completeness, we stipulate that the human analyzing any specific non-constructed computer has no knowledge of any *a priori*, privileged decoding of some of its physical variables as the logical variables of a computational machine.

Many (though not all) non-constructed computers occur in biological systems. Much of the work in the literature on such computers involves the human choosing a way to decode its physical variables so that the resultant computational machine has some desired computational behavior, e.g. so that it is computationally universal. Quintessential examples of such non-constructed computers are found throughout biology, including brains, eusocial insect colonies, and individual cells.

We do not mean to restrict non-constructed computers to biological systems, however. As an example of a non-constructed computer which is not 'biological' in the usual sense, albeit a fanciful example, we could imagine that humanity discovers an alien computer, along with an instruction manual for using it. In this case, no human has constructed the dynamic system, but the 'correct' encoding is known ahead of time to those humans.

This distinction between constructed and non-constructed computers is not as clear-cut as one might wish. In particular, suppose we modified the characteristics of a turbulent flow found in nature to have it implement one computation rather than another. Strictly speaking, this is a constructed computer.

However, because we start with a physical system already found in nature, there are aspects of this system that overlap with those of non-constructed computers.

Indeed, arguably, the classification of computers as either constructed or non-constructed is ultimately anthropomorphic, since it relies on the availability of a preferred decoding map to an external observer (such as a human engineer) ahead of time, rather than on intrinsic dynamic properties of the system alone. However, as already emphasized in this section, this observer dependence is not specific to the constructed versus non-constructed distinction: in fact, a single fixed dynamic system can be viewed as performing infinitely many different computations, depending on the choice of decoding map. Consequently, any attribution of computation necessarily depends on which decoding map is chosen—a choice typically made by a human analyzing the dynamic system.

To illustrate this point, consider an alien encountering both a digital computer and a biological system (e.g. a brain). Depending on the criteria adopted, the alien might classify both systems as 'constructed' (one by humans, the other by evolution), or as 'non-constructed' if no preferred decoding map is accessible to the aliens.

2. Focus of this paper

How should we identify the computation done by a dynamic system when it is non-constructed, and so we do not have a pre-specified way of mapping its states to logical variables? Is there a principled way to identify what computation (or set of computations) is performed by an arbitrary (time-stationary) dynamic system without pre-specifying how we decode its states into logical variables?

In general, we can identify many (often infinite) different computations with the dynamics of any given non-constructed dynamic system [9], even if that dynamic system is not computationally universal (so that the result derived in section 1.1 does not apply). How should we choose one of the many different computations that are consistent with the observed behavior of such a dynamic system, and privilege it as 'the' computation that the system is performing [11]?

As ill-posed as this question is, answering it is a necessary first step to being able to construct a general framework for analyzing what computation(s) are performed by any dynamic system.

In this paper, we illustrate these issues by considering a broad range of non-constructed computers. This focus means that we do not consider nonstandard types of artificial (constructed) computers, like neuromorphic computers [12–14], liquid brain computers, human-constructed analog computers, etc. Just like standard computers, which are digital and typically use von Neumann architectures, the human designer of such non-standard artificial computers determines ahead of time how to identify the variables in the dynamic system with logical variables in an associated computation. This is true of all constructed computers, essentially by definition. Similarly, due to the limits of space, we do not explicitly consider quantum computation [15–17], nor analog computation (see [18, 19] and references therein). We note, though, that the formalism we present in section 4.2 is broad enough to encompass all of these.

As a final comment, we emphasize that we do *not* restrict ourselves to dynamic systems that are either Turing universal and/or that can exhibit uncomputable behavior. (Those specific issues are briefly discussed in section 6 below.) Our interest is in dynamic systems that can be interpreted as implementing any computational machine considered in computer science (CS) theory, even if it is weaker than universal Turing machines (TMs), e.g. any machine from the Chomsky hierarchy.

The structure of the rest of the paper is as follows: we begin in section 3 with a quick review of some of the non-constructed computers that have been investigated in the literature. As mentioned above, in general, many (often infinitely many) different computational machines are implemented by a given non-constructed computer. Accordingly, in section 4.2 we introduce a broadly applicable framework for identifying the set of all possible computational machines that are implemented by any given non-constructed dynamic system.

In section 5 we then briefly illustrate our framework on some of the examples of non-constructed computers found in the literature. Next in section 6 we present a very brief summary of earlier work on TMs implemented by non-constructed computers. We end in section 8 with a discussion of the many instances of non-constructed computers that neither our framework nor any other framework we are aware of fully captures, and so might be quite fruitful topics for future research.

3. Examples of dynamic systems that implement computation

In this section, we discuss several examples of dynamic systems that can be seen as performing computation. The set of examples we present is meant to be illustrative rather than exhaustive; there are many

other examples of dynamic systems that are capable of computation. We start with a summary of a few non-constructed computers:

- **Neuroscience.** While it is common to compare a brain to a computer metaphorically, it has also been suggested that a brain can be viewed as a 'computer' in a more formal sense [20]. In particular, there has been extensive discussion of the hypotheses that the brain is operating near a critical point [21–24], and that it exhibits quasi-critical dynamics [25]. This has led many researchers to apply information-theoretic concepts like predictive coding to analyze neuronal dynamics, see, e.g. [26–29]. Similarly, recent studies have focused on the computational power of the human brain [30].
- **Groups of multiple interacting biological organisms.** There are many instances in biology of *groups* of multiple biological organisms that are commonly viewed as performing computation. One canonical example is eusocial insect colonies. The collective behavior of such colonies, induced by processes like stigmergy [31, 32], the 'waggle dance' of bees [33], etc is often interpreted as performing computation. Indeed, much research has been designing computational algorithms to perform optimization explicitly inspired by stigmergy [34–36].

  There are many other similar examples. A notable one is animal swarms, e.g. of fish, starlings, etc [37], whose joint dynamics has been viewed as performing a computation. These swarms coordinate primarily through visual means. Slime molds, in contrast, are also often viewed as performing computation [38], though visual processing is not involved. Another striking example of naturally occurring dynamic systems, which are often decoded as computing systems, are entire human societies, ranging from hunter-gatherer tribes to modern economies [39, 40].
- **Computations by single cells in multi-cellular organisms.** There are also many examples of *single* biological organisms that are commonly viewed as performing computation, even those lacking a nervous system. One canonical example is genetic regulatory networks [41, 42], which are often formulated as extensions of Boolean circuits, with recurrence allowed. Another example is 'chromatin computation', which involves the feedback system of the expressed genes in a cell nucleus, creating proteins that then change which genes are expressed in that same cell nucleus [43]. Even simple ribosomes, translating RNA into amino acid sequences, have been viewed as a 'simple kind of computer' [44]. See [45] for other examples along these lines.
- **Canonical examples of non-constructed computers in philosophy of science**. There has also been some semi-formal work in the philosophy of science on what criteria should be used to identify the computation done by a given dynamic system. For the most part, this work does not distinguish between constructed and non-constructed computers. It also does not consider the broad range of kinds of (non-constructed) computers discussed above. Specific topics addressed in this literature range from (ontic) pancomputationalism to the role of counterfactuals to the physical Church–Turing thesis. See [9] for a review.

Next, we describe several important classes of dynamic systems that have been used in the literature to investigate the relationship between dynamic systems, broadly construed, and computational machines. Since these dynamic systems are purely abstract, strictly speaking, they are constructed computers. Nonetheless, they are important to understand since they illustrate many of the central issues in how dynamic systems can be seen as performing computation.

- **Systems far from thermodynamic equilibrium**. In the real world, essentially all computers that complete a calculation in finite time must operate far from thermodynamic equilibrium. In the last few decades, a set of powerful theorems concerning the thermodynamics of such systems has been derived by applying recent results from stochastic thermodynamics to analyze idealized computational systems. These results from stochastic thermodynamics include various thermodynamic speed limit theorems [46, 47], thermodynamic uncertainty relations [48–50], and first-passage time entropy production results [51, 52]. All of these results provide formal bounds to the evolution of every physical dynamic system, and so, in particular, these results apply to all physical computational systems, whether evolving stochastically or deterministically [53–56]. The results in this body of work are sometimes called 'stochastic thermodynamics of computation'[4, 57]. Recent work in the stochastic thermodynamics of computation has analyzed thermodynamic costs of TMs [58], computation with circuits [54, 59], and discrete finite automata [60], even when there is absolute irreversibility [53]. Many topics remain open [4].

- **Cellular automata (CA)**. CA are a class of dynamic systems that involve grids (typically infinite, and often one-dimensional) of 'cells', each of which has its own small state space (often just a single bit) [61]. In each (discrete time) step of a CA, all of the cells update their state based on their earlier state and that of the cells in their immediate neighborhood on the grid. Typically, the rule for such updates is the same for all cells in the CA. Therefore, this rule specifies a dynamic system whose state space is the joint state of all of its cells (sometimes called a 'configuration').

  CA can exhibit extremely rich types of behavior even for simple, homogeneous rules. As a result, they have been considered prototypical systems to investigate the possibility of the emergence of artificial life [62, 63]. Adopting a CS paradigm instead, one would like to view any given CA as a computational machine. The idea would be to identify the initial joint state of all the cells as the input to the computational machine, while the subsequent dynamics of the joint state represent the computation itself [64].

  Crucially, though, there is no *a priori* specification of how the configurations of a CA encode the inputs and outputs of a computation (never mind how such configurations at intermediate times encode the intermediate states of a computational machine). In this, CAs are a canonical example of a dynamic system without any interpretation as a computer that we have chosen ahead of time.

  That being said, there has been some quite fruitful work investigating the connection between CA and computation, where one chooses such an encoding ahead of time and then sets the initial configuration accordingly. This line of work stretches back at least to [65]. In that and subsequent work, Wolfram argued that a CA running 'Rule 110' (using his terminology) is Turing complete [66] when the number of cells is countably infinite. This was later formally proven by Cook [67]. The deep relation between TMs, CA, and also both formal systems and Godel's incompleteness theorem was extensively discussed in [68]. This deep relation was more recently used to define a 'biological arrow of time' [69]. The relation between CA and TMs has also been recently studied in the specific case of fungi [70].
- **Symbolic dynamics**. Deterministic dynamics of infinite strings of symbols is sometimes called 'symbolic dynamics'. It has been well studied in CS, since many symbolic dynamics systems provide simplified descriptions of more complex computational processes.

  Indeed, note that the direct definition of a symbolic dynamic system can be seen as a TM with the head removed, so that the dynamics of the infinite tape/symbol sequence does not depend on the state of some such head. Despite this, one can prove that the 'generalized shift map' is computationally universal, i.e. as powerful as arbitrary TMs [71]. (See also [72, 73], which is appearing in this special issue.) The proof of this result was soon followed by closely related studies on topological dynamics [74] and computational universality of arbitrary symbolic dynamics [75].
- **Microstate dynamics of systems whose macrostate dynamics are constructed computers**. There has been some very intriguing research considering the empirically observed low-level dynamics of real-world digital computational systems whose high-level dynamics implement digital programs. A natural research question concerning such systems is whether there is a (almost) bijective map between the 'kind of program' being implemented and some broadly defined set of properties of the associated digital dynamic system. If there is, that raises the question of whether we can extend that bijection to infer what 'programs' are implemented by arbitrary dynamic systems, based on the properties of their dynamics. For some examples, see [76, 77].

4. Universality of emulating computational machines

As described above, our goal in this paper is to investigate non-constructed computers, which are dynamic systems where we must 'infer' the decoding of some of the physical variables of that system into the logical variables of a computational machine, so that we view the dynamic system as implementing that machine. To address this issue in its full generality, we need to formalize precisely what we mean by such 'decoding'. In other words, we need a map φ^{-1} taking (a coarse-graining of) an arbitrary dynamic system A (either in discrete or continuous space and/or time) to a computational machine B such that the dynamics of A can be mapped to the dynamics of B.

To help ground our definitions of the maps back and forth between a computational machine and a dynamic system, in the next subsection we present a proof sketch for the claim in the introduction, that a given dynamic system can be viewed as performing an infinite set of different computations, depending on the particular choice of the decoding map. The following parts of this section then present our full set of definitions, in a completely formal manner.

4.1. There are many dynamic systems that can be viewed as performing an infinite set of computations

Essentially, the claim is established just by defining terms. First, for any universal TM U, write (p,x) for an arbitrary input to U, where p is the specification of an arbitrary TM and x is a bit string in the halting set of p. So we can write $U(p,x)$ for the output tape of U when it reaches a halting state after starting with an input (p,x) in U's halting set, and $U(p,x) = p(x)$, the output of p when it reaches a halting state after starting with input x.

Fact 1. It is well-known that there are many pairs of physically realized dynamic system Ξ, together with an associated UTM U, such that the state space X of Ξ contains two non-overlapping countably infinite subsets A, B with the following properties:

(i) There is a surjective **encoding** map α_A taking the set of all possible (p,x) in the halting set of U to A:
(ii) There is a bijective **decoding map** α_B between the set of all possible $U(p,x)$ and all of B;
(iii) For all pairs (p,x), the first state in B that Ξ reaches after being in state $\alpha_A(x)$ is $\alpha_B(U(p,x)) = \alpha_B(p(x))$.

Suppose we are given such a pair of a dynamic system Ξ and associated UTM U. Let p' be the encoding into the input string of U of some specific (always halting) TM. We wish that TM to be **emulated** by the dynamics of Ξ if we use some appropriate decoding map of the states of Ξ. Formally, this means that there is some countably infinite subset $C \subset X$, some (possibly finite) $D \subset X$, and associated maps α_C and α_D, such that for all bit strings x_0 in the halting set of the TM p', if Ξ is in state $\alpha_C(x_0)$ at some time then the first state that Ξ reaches in D after then is $\alpha_D(p'(x))$.

We can use (Ξ, U) to perform this emulation by taking $C := \{(p,x_0) \in A : p = p'\}$, $\alpha_C(x) := \alpha_A(p',x)$, $D = \{y \in B : \exists x\, \alpha_B(U(p',x)) = y\}$, and for all $x \in D$, $\alpha_D(x) := \alpha_B(U(p',x))$. Since the TM p' is arbitrary, this establishes the claim.

Note that there is no assumption in fact 1 concerning the dynamics of Ξ for any time after it first reaches B. We also make no assumptions about the dynamics of Ξ once it leaves A but before it reaches B. So our definition does not concern the relationship between the step-by-step evolution of Ξ and the step-by-step evolution of p.

In this, the definition in fact 1 is weaker than the definition of a computationally universal dynamic system considered below in section 4.2. However, with some more work, the same result that is proved here can be extended to apply to the dynamic systems considered in the next section section 4.2.

4.2. Formal definition of emulating a computational machine with a dynamic system

In this subsection we both fully formalize and extend the definitions introduced in section 1.

First, we adopt the common definition that a **dynamic system** is any triple (X,T,f) where:

(i) X is a set whose elements we call 'states';
(ii) T is a set of (real-valued) times;
(iii) $f: T \times T \times X \mapsto X$ such that for all $x_t \in X$, for all $t, t', t'' \in T$ such that $t'' > t' > t$:
  (a) $f(t,t'',x_t) = f(t',t'',f(t,t',x_t))$
  (b) $f(t,t,x_t) = x_t$.

The requirement of (iii)a expresses the fact that we can interpret f as the evolution operator of a system whose dynamics is deterministic. In contrast, the requirement of (iii)b simply expresses the fact that the system cannot be in more than one state at once. As shorthand, for all $A \subseteq X$, write $f(.,.,A)$ for the map taking A to the set $\{f(.,.,a) : a \in A\}$. We will say that a dynamic system (X,T,f) is **reversible** if for all $t, t' > t, x \in X$,

$$\nexists x' \neq x : f(t,t',x) = f(t,t',x')\,. \tag{1}$$

Next, we need to define the computation that we wish (X,T,f) to implement/emulate. Such a computation is itself a dynamic system, of course, just evolving over a space of 'computational variables' Y rather than over an arbitrary X. Moreover, in general, we are interested in pairs of a dynamic system (X,T,f) and computation (Y,T,g) such that the dynamics of more than one $x \in X$ will emulate the evolution of a single element in Y. (In other words, the emulation will be coarse-grained.) Accordingly, we say that the dynamic system (Y,T,g) is **emulated** by the dynamic system (X,T,f) if there is a surjective function φ from elements in Y to the elements of a partition of X such that

(iv) for all $t, t' \in T$, $y \in Y$,

$$f(t, t', \varphi(y)) \subseteq \varphi(g(t, t', y)) . \tag{2}$$

The use of $\subseteq$ rather than a strict equality in item iv reflects the fact that we do not require that the initial set $\varphi(y)$ has to evolve into the whole set $\varphi(g(t, t', y))$ under the dynamics, but it can evolve just into a proper subset, still corresponding to the same state $g(t', t', y)$ of the computational machine.

We will abuse notation and write $\varphi^{-1}(x) := y$ for all $x \in \varphi(y)$. In the sequel, we will sometimes refer to the dynamic system (Y, T, g) as a 'computational machine'.

We will sometimes refer to φ^{-1} as the **decoding** function, with φ being the **encoding** function. Reflecting the roles usually played by the two dynamic systems, when the dynamic system (Y, T, g) is time-reversible we will say that it is **logically reversible**, and when the dynamic system (X, T, f) is time-reversible we will say that it is **physically reversible**. In general, logical and physical (ir)reversibility are distinct properties; the former is the property of a computational device, while the latter is the property of a physical system's dynamics. In particular, by different choices of φ, the same dynamical system that is physically reversible can simulate either logically reversible or logically irreversible computations, depending on φ [78–80]. (See [81] and references therein.)

Note that different y get mapped by φ to non-overlapping subsets of X, i.e. to different bins in a coarse-graining of X. Although in general it is not required, in the rest of this paper, we assume that the dynamics given by f over these coarse-grained bins is single-valued. To formalize this, write the coarse-grained bin containing any x as $\mathcal{X}(x)$. So we implicitly assume that for all $x, x' : \mathcal{X}(x) = \mathcal{X}(x')$, and for all $t, t' > t$,

$$\mathcal{X}[f(t, t', x)] = \mathcal{X}[f(t, t', x')] . \tag{3}$$

Combining this requirement with item iv means that we can follow a three-step procedure to determine how g will evolve any $y \in Y$ from t to t'. First, we pick any x in the associated subset $\varphi(y) \subseteq X$. Next we evolve that x from t to t' using f. Then we find the (unique) y' such that $\varphi(y')$ contains that evolved x. We can write this three-step procedure as

$$\varphi^{-1} f(t, t', \widehat{\varphi}(y)) = g(t, t', y) \tag{4}$$

where $\widehat{\varphi}(y)$ is a single-valued function from Y to X that is arbitrary, so long as for all $y \in Y$, $\widehat{\varphi}(y) \in \varphi(y)$.

We emphasize that our definition of emulation is completely general. In particular (Y, T, g) can have super-Turing computational power, or can be strictly weaker than TMs (e.g. a deterministic finite automaton). In particular, there is no specification of some special 'halt states' in X. This ensures that (for example) our definition encompasses dynamic systems like the human brain, that never reach a 'halt state', but rather compute on a continual basis. However, if we wish to consider emulation of a TM, we can simply choose some set of bins in $\mathcal{X}(X)$ that we wish to identify as halt states, and then require that if (X, T, f) ever reaches such a halt state, it never changes afterward.

One notable peculiarity of emulation is how time arises in the two dynamic systems. Suppose the dynamic system (X, T, f) is a real physical process, while (Y, T, g) is some abstract computational machine, e.g. one of the machines in the Chomsky hierarchy. Our definition of emulation implicitly treats 'time' in that computational machine as identical to 'time' in the physical dynamic system. At the cost of more complicated notation, we could extend the definition of emulation to allow the time that f takes to finish its emulation to differ from the time that g takes to perform the original map, i.e. so that the times t, t' on the left-hand side of item iv are not equal to the times t, t' on the right-hand side, but rather are given by a function of those two times on the right-hand side. However, there is no reason to consider such notation in this paper.

Note as well that it is straightforward to modify item iv to allow nondeterministic computational machines (as defined in CS theory), by allowing φ^{-1} to be a many-valued function. This would mean that the sets $\varphi(y), \varphi(y')$ might overlap for certain pairs $y, y' \neq y$. (We could alternatively capture nondeterministic computation by modifying the definition of f to be many-valued.) Similarly, it is straightforward to modify item iv to include stochasticity in g (as in probabilistic TMs, stochastic finite automata, etc).

4.3. Formal definition of the central question of this special issue, and the importance of constraints

We can now formalize the topic of this paper as the following, *central question*:

> *What is the set of all dynamic systems* (Y,T,g) *that are emulated by any given (arbitrary) dynamic system* (X,T,f)*?*

Note that this central question is not the same as the question of whether a *specific* dynamic system (Y,T,g) is emulated by some other *specific* dynamic system (X,T,f). Such variants of our central questions have been investigated in the literature, mostly for the case where (Y,T,g) is a universal Turing machine [65, 71–73, 82–87], or an extension of one [75, 88]. This special case is discussed further below, in section 6.

In general, we (and the other contributors to this focus issue) are primarily interested in these issues when certain constraints are imposed. For example, we might require that $T=\mathbb{Z}$, $Y=\mathbb{B}^*$, and (Y,T,g) is a machine in the Chomsky hierarchy. A second example is where f is an evolving physical system, and we require f to reach a physical state we interpret as a 'halting condition' while using less than some bound on (physical) time. A related, third example is where f is again an evolving physical system, where we might require f to reach a physical state we interpret as a 'halting condition' while using less than some bound on (physical) free energy. (Note that the second and third of these examples differ only in what resource cost is of interest.)

4.4. Constraints on the computational power of the encoding and decoding maps

Another kind of constraint that is very often imposed on our central question, even though it is almost never stated explicitly, is that no non-trivial computation is allowed to be 'hidden' in the map $\widehat{\varphi}$. Intuitively, very often we want to ensure that the dynamics of the computational machine being emulated is not hidden in the encoding function from the initial state of that machine to the initial state of the dynamic system that will emulate it.

An extreme version of this constraint is forbidding the initial state $\widehat{\varphi}(y)$ to already be the final result of the computation, $g(t,t',y)$. After all, if that were the case, it would mean that $f(t,t',\widehat{\varphi}(y))$ only needs to implement the identity map. For similar reasons, we do not want any non-trivial computation to be 'hidden' in the decoding function φ^{-1}.

A natural way to enforce such constraints on the encoding and decoding functions is to bound the time complexity or space complexity (in CSs sense of the terms) of any universal TM that implements those maps $\widehat{\varphi}$ and φ^{-1} [6, 89]. To illustrate this, consider the case where T is the set of all natural numbers. Suppose as well that both X and Y are countably infinite (which, if (X,T,f) is a real classical dynamic system, would mean that X is a coarse-graining of phase space). So the state spaces of both dynamic systems can be represented as (a subset of) $\mathbb{B}^*$, the set of all finite bit strings. This means that both $\widehat{\psi}$ and ψ^{-1} are maps from $\mathbb{B}^*$ into $\mathbb{B}^*$.

In real-world scenarios of this type, we typically want $\widehat{\varphi}$ to be easily computable, e.g. in polynomial time. After all, if it is worse than polynomial time, then we would not be able to initialize the emulating dynamic system (X,T,f) in a timely manner if the initial bit string $y\in Y$ whose evolution we wish to emulate is too long. Indeed, if g is the update function of a UTM that runs in polynomial time for all inputs in its halting set, then emulating that UTM with (X,T,f) would be counterproductive. In other words, if we impose the constraint that $\widehat{\varphi}$ be polynomial time and so polynomial space, it ensures that not much computation is hidden in $\widehat{\varphi}$.

Similarly, again as a practical issue, we typically want φ^{-1} to be easily computable (e.g. in polynomial time), in order to 'read out' the state of the dynamic system f at the time t' as the result of the computation g. If we impose such a constraint, it ensures that not much computation is hidden in φ^{-1} (see [90]). So by imposing these constraints on φ^{-1} and $\widehat{\varphi}$, respectively, we ensure that neither is more computationally powerful than a polynomial computational machine (Y,T,g)[5].

One downside of this approach to enforcing the constraints is that often, if g is more computationally powerful than polynomial time, we would want to allow $\widehat{\varphi}$ and/or φ^{-1} to match g's power, rather than be forced to be weaker. One way to extend the approach to handle this scenario is to only require that $\widehat{\varphi}$ and/or φ^{-1} be (polynomial time) reducible to g. For the case of $\widehat{\varphi}$, this would mean that there exists a polynomial-time TM K mapping all $(t,t'>t,y)$ to a pair $(y^*,t^*>t)$ such that $\widehat{\varphi}(y)=g(t,t^*,y^*)$

[5] As an aside, note that imposing such a constraint on $\widehat{\varphi}$ is closely analogous to the requirement that a reduction (in the CS sense of the term) from one computational machine to another be polynomial time. In CS, though, it is implicitly assumed that there is no need to consider the time complexity of the analog of φ^{-1}, which corresponds to inverting the reduction, to get back the answer to the original decision problem. In fact, inverting functions is the focus of a different body of research in CS theory, involving one-way functions, cryptography, etc [6].

and such that $K(t,t',y)$ completes in a time that is a polynomial function of the length of the binary representation of (t,t',y) [6, 89]. (The case of φ^{-1} would be defined similarly.)

5. Illustrations of the formal definition of dynamic systems emulating computational machines

In this section, we briefly summarize some additional examples of dynamic systems that emulate computers (or seem to), beyond those already presented in section 3. Our goal with these examples is to illustrate our definitions in section 4.2.

In all of these examples, we present an overview of how to establish that the dynamics of the appropriately instantiated physical systems is equivalent to a TM, under the appropriate interpretation of its states in terms of logical variables (i.e. for the appropriate decoding function). With some caveats, described below, these equivalences mean that the associated physical systems are 'Turing-complete', i.e. capable of performing any computation that a TM can.

In order to establish such Turing-completeness, the typical strategy is not to show directly the equivalence between the dynamic system and the TM. Usually, it is more convenient to construct a mapping to a system that is well-known to be Turing-complete (systems like the lambda-calculus, generalized recursive functions, register systems, Post canonical systems, or CA, just to name a few). This is the strategy in the examples mentioned below.

5.1. Billiard ball model of computer

One of the earliest attempts to build a computer that operated purely mechanically was proposed by Fredkin and Toffoli [91]. They introduced the concept of *conservative logic*. Conservative logic is designed to respect both physical and logical reversibility. It is also designed to respect *composition*, which can be related to assumption (iii) of the definition above. Specifically, they considered a model of balls with a fixed non-zero radius initially positioned and moving in a suitable fashion in a properly configured container (i.e. one containing appropriately angled walls). They are mapping that system in such a way that the container configuration represents a specific logically reversible Boolean circuit, while the presence/-absence of a ball in a specific position represents the state of a gate of that circuit. They then showed that, supposing collisions between the balls and between them and the walls of the container are purely elastic, this system could implement any Boolean circuit. In this sense, their billiard ball model is computationally universal [5].

We can identify the quantities in this model of a computer with the quantities in section 4.2 as follows:

- X is an appropriate finite subset of the (classical) phase space of N hard balls, Γ. We denote the coarse-grained space as $\underline{\Gamma}$.
- f is the discrete-time dynamics through Γ assuming they evolve according to frictionless kinetics with perfectly elastic collisions. f also encodes the container that lets the balls elastically reflect the container walls when they hit them.
- Y is the Cartesian product of an integer-valued discrete-time counter and $\mathbb{B}^N$, where $\mathbb{B} := \{0,1\}$ and N is the number of Boolean gates in the Boolean circuit that is being implemented by the billiard ball computer.
- g is specified by the dynamics of the (logically reversible) Boolean circuit, i.e. the (counter-dependent) discrete-time map setting the joint state of an associated subset of N of the binary gates, based on the just-computed values of each gate's parents in the circuit.
- The domain of φ^{-1} is the Cartesian product of $\underline{\Gamma}$ and the value of the counter, i. The function φ^{-1} maps any such element to the joint state of the subset of N of the gates in the Boolean circuit that was just set after the i'th iteration of the gates in the circuit. Note that the subset of the N gates uniquely specifies the ending joint state of the output gates of the circuit.

We assume that the dynamic system (X,T,f) together with φ (and therefore together with $\underline{\Gamma}$) obeys equation (3). At the initial value of the counter, Y includes the input gates, and at the final value of the counter, it includes the output gates.

5.2. Computation with chemical reaction networks

Another example of a physical system that is capable of emulating a computer is a chemical reaction network (CRN), i.e. a finite set of coupled chemical reactions defined stoichiometrically, e.g.

$$A + B \rightarrow C + D.$$

Appropriate CRNs can encode a broad set of computational functions, where the input is the initial concentrations and the output is the final concentrations after the set of reactions equilibrates. It has been shown that non-inhibitory CRNs in which reaction rates can vary arbitrarily over time can encode continuous, piecewise linear functions [92]. On the other hand, inhibitor CRNs (iCRNs) that consist of reactions of the form

$$A + B \overset{/I}{\to} C + D,$$

where the reaction happens if reactants$A$ and B are present and the inhibitor I is absent, can compute any computable function $f: \mathbb{N} \to \mathbb{N}$ [93]. This is shown by mapping the iCRNs to the register machines, which are known to be Turing-complete [94].

We can identify the quantities in this model of a computer with the quantities in section 4.2 as follows:

- X is the set of concentrations of the chemical compounds used in the CRN; the initial concentrations correspond to the input variables, and the final (i.e. the equilibrium) concentrations correspond to the output variables.
- f is the chemical reaction network, i.e. the list of reactions that are possible
- Y is the set of registers R_i in the register model
- g is the set of instructions in the register model (e.g. the increment function `INC` or decrement function `DEC`)
- φ describes the mapping from the concentration operations through the iCRN to the set of instructions (i.e. the algorithm) in the register machine

5.3. Fluid computers

We end with a discussion of some types of dynamic systems that do *not* emulate TMs, in the formal sense of our definitions, even though they do have a certain kind of 'Turing-completeness'.

Ever since [71], several researchers have raised the question of whether fluid flows are capable of performing (universal) computation. Tao suggested [95] that there might be a connection between potential blow-ups of Navier–Stokes equations, Turing completeness, and such fluid computation. In a recent paper [72], the authors constructed a TM from a dynamic system described by the Euler flow on Riemann $\mathbb{S}^3$ by identifying the flows with generalized shift maps that are equivalent ('conjugate') to a TM. The way they did this can be summarized as follows. First, they construct a compact 3-dimensional manifold where Euler flows are defined, i.e. where the set of stationary vector fields $\mathbf{u}(x)$ that fulfill the following conditions

$$\nabla \mathbf{u} = 0 \quad \text{(incompressibility)}$$

$$(\mathbf{u} \cdot \nabla)\,\mathbf{u} = -\nabla p \quad \text{(Euler equation)}$$

is well-defined. The streamlines (also called *trajectories*) of $\mathbf{u}$ $x_{\mathbf{u}}$ ($\dot{x}_{\mathbf{u}} = \mathbf{u}(x_{\mathbf{u}})$) are identified with time evolution on the manifold. The conjugacy with TMs is established through showing the equivalence between the Euler flow and the Generalized shift map, which was shown to be conjugate to a TM in earlier work [71].

A small variant of this group of authors also showed the existence of 'Turing-complete' flows in other compact manifolds [96] as well as in Euclidean spaces [97]. Furthermore, in a series of recent studies, the authors show that the flow on a smooth bordism of a vector field with good local properties can simulate any Turing-computable function [73]. Finally, they construct Turing-complete steady flows of Navier–Stokes fields [98]. A recent discussion can be found in [99].

In a recent study [100], this and related approaches to emulating computation in dynamic systems were summarized with the following definition of (what the authors call) 'Turing-complete':

> 'A dynamic system X on the topological space M is Turing complete if there exists a universal TM T_U such that for each initial configuration c of T_U, there exists a (computable) point $p_c \in M$ and a (computable) open set $U_c \subset M$ such that T_U halts with input c if and only if the positive trajectory of X through p_c intersects U_c.'

Although dynamic systems that have this 'fluid computer Turing-completeness' (FCTC) are similar in some aspects to the dynamic systems that emulate UTMs (in the formal sense defined in section 4), there are some important differences.

Often, we are not interested in whether a given TM halts for a given input. Rather, we want to know the *output* of a TM when it does halt, for such a given input. To use an FCTC to decide even just whether a TM T that has input I halts with *some specific* output O, one needs to encode I and O both in p_c. Given some such encoding, one could use a countably infinite set of FCTCs, each run on a different p_c, to perform the desired computation of the actual output O. For example, one could do this using the 'dovetailing' procedure of CS theory. However, this procedure would require a second dynamic system to sift among that infinite set of FCTCs, watching which has halted when, and what output O is encoded in the p_c of that FCTC if and when it does halt.[6]) No such dynamic system is considered in [72].

Note as well that even if the definition of an FCTC were extended somehow to have it compute the output string when a given TM with a given input halts, FCTCs would still only implement the partial function of that TM. They provide no information about the full dynamics of that TM through its state space of instantaneous descriptions (IDs), as it evolves from one iteration of its update function to the next. In other words, they do not construct an analog of the map φ central to our approach, taking an arbitrary state of the TM to a subset of X. Nor does the definition of an FCTC involve a set T of times, like those considered in our approach. These differences between FCTCs and the kinds of systems we consider above reflect the fact that the goals in the investigations of FCTCs are different from ours.

6. Uncomputability in dynamic systems

Many dynamic systems (X, T, f) have been explicitly shown to have properties that are uncomputable, in that no TM that is guaranteed to halt for all inputs can compute those properties ahead of time [87]. Recent work has discussed examples of such systems. Bausch *et al* [101] proved that for a specific Hamiltonian of a many-body quantum system, the task of determining the phase diagram is generally uncomputable. This result is based on the fact that the problem of deciding whether a many-body quantum system has a spectral gap is at least as hard as solving the Halting problem, and therefore undecidable (i.e. the bit of whether there is a spectral gap is uncomputable) [86]. Similarly, Shiraishi and Matsumoto [85] showed that for a general quantum many-body system, the question of whether the system thermalizes is undecidable. Undecidability has not only been established for quantum systems. For instance, Gonda *et al* [102] discussed whether a general spin model is equivalent to a TM. Additionally, Purcell *et al* [103] created a family of Hamiltonians that has a guaranteed single phase transition. Yet, the location of this phase transition is uncomputable since it is determined by the value of Chaitin's constant, which is known to be uncomputable.

These kinds of results have motivated several authors to suggest that it might be useful to study physical dynamic systems with more computational power than TMs [104, 105]. These systems are typically called *super*-Turing capable systems in the literature.

The concept of dynamic systems with super-Turing capability leads us to carefully consider the original form of the Church–Turing thesis. As recently discussed in [106], there are at least three forms of the Church–Turing thesis. The strongest version called *Super-bold physical Church–Turing thesis* or *total physical computability thesis*, assumes that every physical aspect of any real-world system must be Turing computable. While some of the aforementioned examples might point to the fact that this thesis might be too strong, it is open to debate whether the uncomputability problem is not just a mathematical artifact of the underlying theories (e.g. quantum mechanics) and whether it is rather not a constraint that the underlying physical theories should follow.

On the other hand, there are at least two weaker versions of the Church–Turing thesis that are not so restrictive. The first one, the *Church–Turing–Deutsch–Wolfram* thesis, proposes that every physical system can be simulated on a TM. An even more modest version, called the *Physical computation thesis*, proposes that every function computed by any physical computing system is Turing-computable.

To date, it is not known which (or even whether) any of the forms of the Church–Turing thesis is correct. However, recent discussions suggest that the Church–Turing thesis should not be understood primarily as a claim about the intrinsic computability or super-computability of physical systems, but

[6] In a dovetailing procedure, one would run the dynamic system simultaneously, in a staggered manner, for more and more output strings O, so that if the TM T halts with input I, so would the dovetailed instances of the dynamic system, and furthermore, the instance of the dynamic system that halts would be the one specifying the precise output O produced by T on input I. (One might think it would suffice to just run a single FCTC iteratively, for every possible output O. However, this would have the problem that for some given T, I, where $T(I) = O$, the dynamic system never halts for some output O' that it is run on (in the sense that it goes through the point $U_{I,O,T}$) before it is ever run on the correct output, O. The dovetailing procedure avoids this problem by running iteratively more and more instances of the dynamic system at the same time, just for different inputs, in a staggered fashion [7]. This dovetailing approach will result in one of the instances of the dynamic system halting with the correct output O, and no instances will halt with an incorrect output).

rather as an epistemic statement: namely, about whether we could ever confirm that a dynamic system we encounter computes functions beyond the Turing limit. From this perspective, even if a physical system possessed such capabilities, our inability to unambiguously identify and verify the corresponding decoding map would prevent us from operationally establishing this fact. This viewpoint reflects the approach taken in this paper, which emphasizes that claims about computation are necessarily relative to the observer's access to and justification for a particular decoding map.

7. Value of computation

For many dynamic systems, both constructed and non-constructed, it might not be particularly important to identify the precise computation done by the dynamic system, nor the amount of computation that it does. Rather, it is the **value of computation** done by such systems that is their most important property. Such a value of computation might be the most important quantity concerning the future evolution of the system. A typical example is (biological) evolutionary systems, in which the value of computation can be defined by the future dynamics of one or more fitness functions.

To illustrate this in more detail, consider an agent that has some initial information about its environment, and which uses that information to *compute* the best action to take on its environment, in order to increase the value of its 'viability'. For example, this viability may be the ability of the agent to extract free energy from its environment, i.e. to adaptively respond to environmental perturbations that it senses, thereby maximizing long-term reproductive fitness. The better the computation the agent does, the higher its viability will be after this subsequent interaction with its environment. Some preliminary work has been done in [107, 108].

To make this concrete, suppose we have three time-steps, $t = \{t_1, t_2, t_3\}$, and an agent who comprises two variables, a memory variable, X^m, and an action variable, X^a. We represent the agent's variables as $X = \{X^m, X^a\}$. For simplicity, we represent the state of the environment as a single variable, Y. The joint system's dynamics proceed as follows:

(i) At time $t = t_1$, there is some statistical coupling between X^m and Y, e.g. reflecting earlier observations by the agent of variables in the environment.
(ii) During the time interval $t \in (t_1, t_2]$, X and Y evolve independently of one another, i.e. $P(y_t, x_t | y_{t_1}, x_{t_1}) = P(y_t | y_{t_1}) P(x_t | x_{t_1})$. We can also assume that X^a does not change during that evolution. We identify the evolution of the agent's joint memory state during this time as its performing a 'computation', based on the initial value of its memory at the start of the interval.
(iii) At the end of this interval, at $t = t_2$, there is a conditional distribution $P(X^a | X^m)$ that uses the state of the agent's memory to set the value of their action variable.
(iv) Then, during $t \in (t_2, t_3]$, the agent's variable X^a and environment are coupled via some (exogenously specified) interaction Hamiltonian $H_{\text{int}}(x, y)$ that was set to 0 before the interval
(v) At $t = t_3$, that interaction Hamiltonian is set to 0, i.e. it is removed. For completeness, we can assume that X^m does not change during that time interval $t \in (t_2, t_3]$.
(vi) At this point, the entire cycle can repeat, perhaps after an intervening interval in which the statistical coupling between X^m and Y is reset.

As in the semantic value of information framework [109], we are interested in the value of a viability function concerning the distribution $P(x_{t_3})$. Examples of that function include the KL divergence from $P(x_{t_3})$ to the stationary state of X under a rate matrix governing its dynamics, or the KL divergence to the Boltzmann distribution of the agent [109, 110]. (Note the similarity of this setup to closed-loop control, in control theory, just with unusual objective functions).

In other situations, other kinds of viability are more directly of interest. In particular, often the relevant type of viability involves *embodiment constraints*. As an example, often we are interested in the minimal amount of thermodynamic work required by the agent to go from the distribution over its possible states (X_m, X_a) at t_3 to some desired target distribution over those states, before the cycle repeats. In this case, the viability function would be (negative of) the amount of work expended to effect that change in the distribution. Crucially though, the agent must implement that change in distributions while subject to some set of constraints on rate matrices and/or Hamiltonians governing the dynamics of X, constraints which reflect limitations of the agent. Those are the embodiment constraints.

As a variant, in some situations it makes sense to consider viability functions that do not involve the amount of work required to go from one ending state distribution to another. For example, if the agents under consideration are biological organisms, the viability function could be the difference in the beginning and ending distributions of some particular non-energetic resource (like carbon, or phosphorus,

or some such) rather than the amount of work. Such viability functions can be of interest either where there are embodiment constraints or not, depending on the precise scenario that is being modeled.

In all of these scenarios, we would identify the dynamics from $X^m_{t_1}$ to $X^m_{t_2}$ as emulating some specific computation, i.e. that dynamics is the computation performed by the agent. The precise form of the dynamics over X^m would specify the precise computational machine being emulated by the agent. The value of that computation would quantify the 'power' or 'complexity' of that computation.

8. Future work

There are many avenues to be explored for future research. Some were discussed above, in the main text. Here we list a few others of particular interest.

- **Computers that interact with input and output systems.**
  All constructed computers crucially rely on there being at least two additional dynamic systems that they interact with:
  (I) An input system to determine the initial state of the computer and/or an input (or stream of inputs) into that computer;
  (O) An output system to observe the outputs of the computer.
  (The analysis of computers that receive streams of inputs was pioneered by Karp; see [111].)

  Unless it is accompanied by these extra systems, no human would be able to actually *use* a particular constructed computer. However, just because it is accompanied by such input and output systems does not ensure that a given constructed computer is usable. For example, an input system that requires fixing an infinite number of digits in an arbitrary real number would be ruled out, as (tautologically?) beyond human capability to construct.

  How does the problem of identifying computation in real-world dynamic systems change if we expand it to involve triples of dynamic systems, including both (usable) input and output systems, in addition to a computer system that they are coupled with? How does the problem change if we combine (I) and (O) into a single system, an external environment that interacts with the computer's outputs and determines its inputs?
- **Dynamic systems as optimizers of an objective function.**
  One approach to inferring what computation is done by an arbitrary non-constructed dynamic system relies on knowing (or assuming) an objective function that the dynamic system optimizes. In particular, some research in neurobiology has imposed specific tasks on a biological system and then tried to use the system's (neurological) response to those tasks to infer the dynamic system's computational algorithm for achieving that task. This approach implicitly presupposes that the dynamics of the biological system are optimized to perform well at the specific task that the researcher is able to impose on the system. For example, if the biological system is an organism with a central nervous system and a large brain, this approach has relied on presupposing that the brain can perform optimally at the task of recording a memory [112], or at the task of decomposing the form of a sensory stimulus after the brain has processed it into a form that matches the structure of the stimulus [113].

  However, in the vast majority of non-constructed dynamic systems, we have no such objective function we can rely on to identify the computation performed by the system. Even in the context of neurobiology, often the fundamental goal an organism is addressing can remain elusive, never mind what computation they might be using to achieve such a goal.

  If we do restrict ourselves to systems where there is a clear objective function, what are the best methods for inferring from the change in the dynamics of the system when the task is imposed, exactly what computational algorithm is used to achieve the task?
- **Quantifying the amount of computation emulated by a dynamic system rather than the precise computation.**
  A perhaps simpler issue than identifying the precise computation emulated by a dynamic system (X, T, f) would be to identify the 'amount of computation' done by that system. In general, this would mean we do not need to consider an explicitly emulated dynamic system (Y, T, g), or a map taking the dynamics given by g to the dynamics given by f [114].

  As an illustration of this approach, we should mention the rich theory that quantifies the 'randomness' of an infinite string using the machinery of algorithmic randomness (i.e. prefix-free Kolmogorov complexity). The idea would be to suppose that any given observed finite string ω of length n is just the first n symbols of an infinite string, $\hat{\omega}$. The supposition is that the infinite string $\hat{\omega}$ is being generated by some computational system, with noise added (e.g. due to observational

limitations). The idea is to examine the substrings in ω to estimate both the rate of 'how much computation' was being done by the computational system, and the rate of noise being added to generate $\hat{\omega}$.

For example, in the limit that $n \to \infty$, *if the string were random in the Martin-Löf sense*, then its randomness deficiency would approach a constant, i.e. the difference between the (prefix) Kolmogorov complexity of the first n symbols in the strings and the length of the string would approach a constant as $n \to \infty$ [7, 115, 116]. To generate any such maximally random string, a (prefix) universal TM would require the maximal possible (rate of)'amount of computation'.

To instead measure the (rate of) computation performed by a dynamic system that does *not* saturate the upper bound, it is natural to hypothesize that the Kolmogorov complexity K of the string ω of length n obeys

$$K(\omega) = \alpha n + c + O(1) \tag{5}$$

for some constants α, c. (The constant α is sometimes called the 'Kolmogorov rate' in the literature.) We could then suppose that α quantifies the fraction of the amount of computation in generating ω, i.e. the amount not due to noise.

To use this idea in practice, we would want to estimate α from our data, i.e. by examining substrings of our single finite string ω. In turn, to make such an estimate, we would need to use a bound on Kolmogorov complexity, since that quantity is uncomputable. Standard choices for an (upper) bound are Lempel-Ziv codelength [117], along with its many variants. (Loosely related approaches have been used to estimate the complexity of biological systems [118], CA [119] or neuronal spike trains [120].)

There are many variations of this approach that might be worth pursuing. Some would involve considering the rate of the conditional Kolmogorov complexity of the most recent symbol in the string, conditioned on the preceding substring as the string length grows. One might even consider quantities like the entropy rate [121] rather than the Kolmogorov rate, especially in light of their tight relationship in the limit of large strings. Of course, yet another related set of approaches could be applied if our data consisted of multiple finite strings, rather than a single one, so that we would focus on a distribution over strings and the associated expectation of the amount of computation.

For example, the fluid computers fit into this class of systems, for which one is able to estimate the complexity of its dynamics. For the recent discussion about this connection, see recent studies [99, 100].

- **Higher-order Markov chain process and amount of computation.**
  As another example of how to quantify the amount of computation done by a dynamic system, note that any real-world computer has lots of memory, reflecting (aspects of its) previous state. This means that the variables in the computer's core evolve in a way that is not first-order Markov. To illustrate this, suppose we look at only the flash RAM of a constructed computer, without considering what's also stored on the disk, the dynamics of that flash RAM is a higher-order Markov process, or equivalently, a hidden Markov model (HMM).

  This suggests that we measure 'how much computation' a given dynamic system Z_t is doing by looking at the order of the Markov chain driving that dynamics, and/or the number of hidden states in a first-order HMM that is driving that dynamics Z_t. There are many ways we could do this. For example, we could consider the maximal delay τ such that the mutual information $I(Z_t; Z_{t-1}, \ldots, Z_{t-\tau}) > \epsilon$ for some prefixed threshold ϵ. Alternatively, we could take the maximal τ such that $I(Z_t; Z_{t-\tau}) > \epsilon$. It is not *a priori* clear which approach is more beneficial, see [122] for some preliminary work along these lines. Alternatively, one can consider a delay-embedding of the process Z_t and estimate the minimal embedding dimension necessary for the process to be linear. (Cf., Takens' theorem [123].)
- **Hierarchical structure of non-constructed computers.**
  Many dynamic systems found in nature that we view as performing computation are physically distributed (often with a hierarchical, modular structure). Can we gain insight into the dynamic behavior of spatially distributed, heterogeneous dynamic systems in general by reformulating them in a first-principles way as performing computation?
- **Dynamic systems as models of continual computers.**
  Naturally occurring dynamic systems that we view as performing computation are very different from standard one-shot computers like those performed by the machines in the Chomsky hierarchy, by machines like CAs, by systems implementing symbolic dynamics, etc. Such one-shot computers receive a single set of one or more inputs at a given time and then compute the associated output(s). In

contrast, naturally occurring systems like the brain are *continual computational systems* — they are getting new inputs continually, at random times, each input requiring a different computational task, and those tasks overlap with one another in time. Physically, these continual computational systems are not just open to the outside world once at initialization. Rather, they are always open, always being perturbed. They are 'embodied' in the real world [111, 124, 125]. Does it help us to understand how arbitrary real-world dynamic systems should be viewed as systems performing computations if we restrict attention to such open, embodied systems?

- **Collective computation.**
  Suppose one has a model of the dynamics of a system. To quantify 'collective computation, focusing on the collective component specifically, one can see how much the dynamics change if we sever all communication among the subsystems. This quantifies the *collective* aspect of the computation, per se. It does not say what it is that's being computed, of course.

These are just a few of the many open problems regarding the theory of non-constructed computers. Many of these problems are connected with the fact that computation is closely related and absolutely crucial for all living systems. Without computation, no living systems could exist. However, as we discussed above, many other dynamic systems are capable of computation. We believe that solving some of the aforementioned issues will be beneficial for understanding both living and non-living non-constructed computers.

Data availability statement

No data were generated during the study. The data that support the findings of this study are available upon reasonable request from the authors.

Acknowledgment

D H W would like to thank the Santa Fe Institute for support. J K acknowledges support from the Austrian Science Fund (FWF) under Grants No. 10.55776/P34994 and EFP5 ReMASS, funding from the Austrian Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation, and Technology under GZ 2023-0.841.266, through the Postdoc Program for Complexity Science and Data Competence.

Author contributions

David H Wolpert 0000-0003-3105-2869
Conceptualization (equal), Writing – original draft (equal), Writing – review & editing (equal)

Jan Korbel 0000-0002-5371-5320
Conceptualization (equal), Writing – original draft (equal), Writing – review & editing (equal)

References

[1] Navlakha S and Bar-Joseph Z 2011 Algorithms in nature: the convergence of systems biology and computational thinking *Mol. Syst. Biol.* **7** 546
[2] Haken H 2012 *Computational Systems—Natural and Artificial: Proc. Int. Symp. on Synergetics at Schloß Elmau (Bavaria, 4 May–9 May 1987)* vol 38 (Springer)
[3] Chu D, Prokopenko M, Christian J Ray J 2018 Computation by natural systems *Interface Focus* **8** 20180058
[4] Wolpert D H *et al* 2024 Is stochastic thermodynamics the key to understanding the energy costs of computation? *Proc. Natl Acad. Sci.* **121** e2321112121
[5] Sipser M 1996 Introduction to the theory of computation *ACM Sigact News* **27** 27–29
[6] Arora S and Barak B 2009 *Computational Complexity: a Modern Approach* (Cambridge University Press)
[7] Ming Li and Vitányi P *et al* 2008 *An Introduction to Kolmogorov Complexity and its Applications* vol 3 (Springer)
[8] Meijers M, Ito S and Wolde P R 2021 Behavior of information flow near criticality *Phys. Rev. E* **103** L010102
[9] Piccinini G and Maley C 2021 Computation in physical systems *Stanford Encyclopedia of Philosophy*
[10] Cotler J and Rezchikov S 2024 Computational dynamical systems *2024 IEEE 65th Annual Symp. on Foundations of Computer Science (FOCS) (Chicago, IL, USA)* pp 166–202
[11] Urai A E, Doiron B, Leifer A M and Churchland A K 2022 Large-scale neural recordings call for new insights to link brain and behavior *Nat. Neurosci.* **25** 11–19
[12] Furber S 2016 Large-scale neuromorphic computing systems *J. Neural Eng.* **13** 051001
[13] Schuman C D, Kulkarni S R, Parsa M, Parker Mitchell J, Date P and Kay B 2022 Opportunities for neuromorphic computing algorithms and applications *Nat. Comput. Sci.* **2** 10–19
[14] Marković D, Mizrahi A, Querlioz D and Grollier J 2020 Physics for neuromorphic computing *Nat. Rev. Phys.* **2** 499–510

[15] Deutsch D and Jozsa R 1992 Rapid solution of problems by quantum computation *Proc. R. Soc. A* **439** 553–8
[16] Feynman R P 1986 Quantum mechanical computers *Found. Phys.* **16** 507–31
[17] Nielsen M A, Dowling M R, Mile G and Doherty A C 2006 Quantum computation as geometry *Science* **311** 1133–5
[18] Siegelmann H T 1995 Computation beyond the Turing limit *Science* **268** 545–8
[19] Siegelmann H T and Fishman S 1998 Analog computation with dynamical systems *Physica D* **120** 214–35
[20] Maley C J 2022 How (and why) to think that the brain is literally a computer *Front. Comput. Sci.* **4** 970396
[21] Chialvo D R 2010 Emergent complex neural dynamics *Nat. Phys.* **6** 744–50
[22] Hesse J and Gross T 2014 Self-organized criticality as a fundamental property of neural systems *Front. Syst. Neurosci.* **8** 166
[23] Shew W L, Yang H, Petermann T, Roy R and Plenz D 2009 Neuronal avalanches imply maximum dynamic range in cortical networks at criticality *J. Neurosci.* **29** 15595–600
[24] Brochini L, de Andrade Costa A, Abadi M, Roque A C, Stolfi J and Kinouchi O 2016 Phase transitions and self-organized criticality in networks of stochastic spiking neurons *Sci. Rep.* **6** 35831
[25] Fosque L J, Williams-García R V, Beggs J M and Ortiz G 2021 Evidence for quasicritical brain dynamics *Phys. Rev. Lett.* **126** 098101
[26] Clark A 2013 Whatever next? predictive brains, situated agents and the future of cognitive science *Behav. Brain Sci.* **36** 181–204
[27] Shiffrin R M, Bassett D S, Kriegeskorte N and Tenenbaum J B 2020 The brain produces mind by modeling *Proc. Natl Acad. Sci.* **117** 29299–301
[28] Shain C, Blank I A, van Schijndel M, Schuler W and Fedorenko E 2020 fMRI reveals language-specific predictive coding during naturalistic sentence comprehension *Neuropsychologia* **138** 107307
[29] Chalk M, Marre O and Tkačik G 2018 Toward a unified theory of efficient, predictive and sparse coding *Proc. Natl Acad. Sci.* **115** 186–91
[30] Gebicke-Haerter P J 2023 The computational power of the human brain *Front. Cell. Neurosci.* **17** 1220030
[31] Khuong A, Gautrais J, Perna A, Sbaï C, Combe M, Kuntz P, Jost C and Theraulaz G 2016 Stigmergic construction and topochemical information shape ant nest architecture *Proc. Natl Acad. Sci.* **113** 1303–8
[32] Heylighen F 2016 Stigmergy as a universal coordination mechanism I: definition and components *Cognitive Systems Research* vol 38 Special Issue of Cognitive Systems Research – Human-Human Stigmergy pp 4–13
[33] Dong S, Lin T, Nieh J C and Tan K 2023 Social signal learning of the waggle dance in honey bees *Science* **379** 1015–8
[34] Salman M, Garzón Ramos D and Birattari M 2024 Automatic design of stigmergy-based behaviours for robot swarms *Commun. Eng.* **3** 30
[35] Nichol P B 2024 Why is stigmergy a good platform for swarm intelligence? (available at: https://www.cio.com/article/234906/why-is-stigmergy-a-good-platform-for-swarm-intelligence.html) (Accessed 1 July 2025)
[36] Theraulaz G and Bonabeau E 1999 A brief history of stigmergy *Artif. Life* **5** 97–116
[37] Ballerini M *et al* 2008 Empirical investigation of starling flocks: a benchmark study in collective animal behaviour *Animal Behav.* **76** 201–15
[38] Adamatzky A 2015 Slime mould computing *Int. J. Gen. Syst.* **44** 277–8
[39] Wolpert D H and Harper K 2025 The computational power of a human society: a new model of social evolution (arXiv:2408.08861)
[40] Hayek F A 2013 The use of knowledge in society *Modern Understandings of Liberty and Property* (Routledge) pp 27–38
[41] Daniels B C, Kim H, Moore D, Zhou S, Smith H B, Karas B, Kauffman S A and Walker S I 2018 Criticality distinguishes the ensemble of biological regulatory networks *Phys. Rev. Lett.* **121** 138102
[42] Davidson E H 2010 Emerging properties of animal gene regulatory networks *Nature* **468** 911–20
[43] Prohaska S J, Stadler P F and Krakauer D C 2010 Innovation in gene regulation: the case of chromatin computation *J. Theor. Biol.* **265** 27–44
[44] Kempes C P, Wolpert D, Cohen Z and Pérez-Mercader J 2017 The thermodynamic efficiency of computations made in cells across the range of life *Phil. Trans. R. Soc. A* **375** 20160343
[45] Nicolau D V, Lard M, Korten T, van Delft F C M J M, Persson M, Bengtsson E, Månsson A, Diez S, Linke H and Nicolau D V 2016 Parallel computation with molecular-motor-propelled agents in nanofabricated networks *Proc. Natl Acad. Sci.* **113** 2591–6
[46] Shiraishi N, Funo K and Saito K 2018 Speed limit for classical stochastic processes *Phys. Rev. Lett.* **121** 070601
[47] Vu T V and Saito K 2023 Topological speed limit *Phys. Rev. Lett.* **130** 010402
[48] Barato A C and Seifert U 2015 Thermodynamic uncertainty relation for biomolecular processes *Phys. Rev. Lett.* **114** 158101
[49] Horowitz J M and Gingrich T R 2020 Thermodynamic uncertainty relations constrain non-equilibrium fluctuations *Nat. Phys.* **16** 15–20
[50] Vu T V *et al* 2022 Unified thermodynamic–kinetic uncertainty relation *J. Phys. A: Math. Theor.* **55** 405004
[51] Gingrich T R and Horowitz J M 2017 Fundamental bounds on first passage time fluctuations for currents *Phys. Rev. Lett.* **119** 170601
[52] Falasco G and Esposito M 2020 Dissipation-time uncertainty relation *Phys. Rev. Lett.* **125** 120604
[53] Manzano G, Kardeş G, Roldán Edgar and Wolpert D H 2024 Thermodynamics of computations with absolute irreversibility, unidirectional transitions and stochastic computation times *Phys. Rev. X* **14** 021026
[54] Yadav A, Yousef M and Wolpert D 2025 Minimal thermodynamic cost of computing with circuits (arXiv:2504.04031)
[55] Yadav A, Caravelli F and Wolpert D 2024 Mismatch cost of computing: from circuits to algorithms (arXiv:2411.16088)
[56] Yadav A and Wolpert D 2024 Minimal thermodynamic cost of communication (arXiv:2410.14920)
[57] Wolpert D H 2019 The stochastic thermodynamics of computation *J. Phys. A: Math. Theor.* **52** 193001
[58] Kolchinsky A and Wolpert D H 2020 Thermodynamic costs of Turing machines *Phys. Rev. Res.* **2** 033312
[59] Wolpert D H and Kolchinsky A 2020 Thermodynamics of computing with circuits *New J. Phys.* **22** 063047
[60] Ouldridge T E and Wolpert D H 2023 Thermodynamics of deterministic finite automata operating locally and periodically *New J. Phys.* **25** 123013
[61] Neumann J V 1966 Theory of self-reproducing automata ed A W Burks (University of Illinois Press)
[62] Langton C G 1986 Studying artificial life with cellular automata *Physica D* **22** 120–49
[63] Johnston N and Greene D 2022 *Conway's Game of Life: Mathematics and Construction* (Nathaniel Johnston)
[64] Greif H, Kubiak A P and Stacewicz P 2024 Selection, growth and form. Turing's two biological paths towards intelligent machinery *Stud. Hist. Phil. Sci.* **106** 126–35
[65] Wolfram S 1984 Cellular automata as models of complexity *Nature* **311** 419–24
[66] Wolfram S 1984 Universality and complexity in cellular automata *Physica D* **10** 1–35

[67] Cook M *et al* 2004 Universality in elementary cellular automata *Complex Syst.* **15** 1–40
[68] Prokopenko M, Harré M, Lizier J, Boschetti F, Peppas P and Kauffman S 2019 Self-referential basis of undecidable dynamics: From the liar paradox and the halting problem to the edge of chaos *Phys. Life Rev.* **31** 134–56
[69] Prokopenko M, Davies P C W, Harré M, Heisler M G, Kuncic Z, Lewis G F, Livson O, Lizier J T and Rosas F E 2025 Biological arrow of time: emergence of tangled information hierarchies and self-modelling dynamics *J. Phys. Complex.* **6** 015006
[70] Schumann A, Adamatzky A, Król J and Goles E 2024 Fungi as turing automata with oracles *R. Soc. Open Sci.* **11** 240768
[71] Moore C 1991 Generalized shifts: unpredictability and undecidability in dynamical systems *Nonlinearity* **4** 199
[72] Cardona R, Miranda E, Peralta-Salas D and Presas F 2021 Constructing turing complete euler flows in dimension 3 *Proc. Natl Acad. Sci.* **118** e2026818118
[73] González-Prieto A, Miranda E and Peralta-Salas D 2025 Topological Kleene field theories: a new model of computation (arXiv:2503.16100)
[74] Kurka P 1997 On topological dynamics of Turing machines *Theor. Comput. Sci.* **174** 203–16
[75] Delvenne J-C, Kurka P and Blondel V D 2005 Computational universality in symbolic dynamical systems *Machines, Computations and Universality: 4th Int. Conf., MCU 2004 (Saint Petersburg, Russia, 21 September–24 September 2004)* (*Revised Selected Papers* vol 4) (Springer) pp 104–15
[76] Garland J and Bradley E 2011 Predicting computer performance dynamics *Int. Symp. on Intelligent Data Analysis* (Springer) pp 173–84
[77] Alexander Z, Mytkowicz T, Diwan A and Bradley E 2010 Measurement and dynamical analysis of computer performance data *Advances in Intelligent Data Analysis IX* ed P R Cohen, N M Adams and M R Berthold (Springer) pp 18–29
[78] Sagawa T 2014 Thermodynamic and logical reversibilities revisited *J. Stat. Mech.* 03025
[79] Landauer R 1961 Irreversibility and heat generation in the computing process *IBM J. Res. Dev.* **5** 183–91
[80] Bennett C H 1982 The thermodynamics of computation-a review *Int. J. Theor. Phys.* **21** 905–40
[81] Piccinini G and Maley C 2010 Computation in physical systems
[82] Moore C 1990 Unpredictability and undecidability in dynamical systems *Phys. Rev. Lett.* **64** 2354
[83] Lloyd S 1992 Any nonlinear gate, with linear gates, suffices for computation *Phys. Lett. A* **167** 255–60
[84] Lloyd S 2006 *Programming the Universe: a Quantum Computer Scientist Takes on the Cosmos* (Vintage)
[85] Shiraishi N and Matsumoto K 2021 Undecidability in quantum thermalization *Nat. Commun.* **12** 1–7
[86] Cubitt T S, Perez-Garcia D and Wolf M M 2015 Undecidability of the spectral gap *Nature* **528** 207–11
[87] Pour-El M B and Richards I 1982 Noncomputability in models of physical phenomena *Int. J. Theor. Phys.* **21** 553–5
[88] Davis M 1956 A note on universal Turing machines *Automata Studies* vol 34 pp 167–75
[89] Sipser M 2012 *Introduction to the Theory of Computation. Cengage Learning* International edn
[90] Delvenne J-C 2009 What is a universal computing machine? *Appl. Math. Comput.* **215** 1368–74
[91] Fredkin E and Toffoli T 1982 Conservative logic *Int. J. Theor. Phys.* **21** 219–53
[92] Chen H-L, Doty D, Reeves W and Soloveichik D 2023 Rate-independent computation in continuous chemical reaction networks *J. ACM* **70** 1–61
[93] Calabrese K and Doty D 2024 Rate-independent continuous inhibitory chemical reaction networks are turing-universal *Unconventional Computation and Natural Computation* ed D-J Cho and J Kim (Springer) pp 104–18
[94] Minsky M L 1967 *Computation* (Prentice-Hall)
[95] Tao T 2019 Searching for singularities in the Navier–Stokes equations *Nat. Rev. Phys.* **1** 418–9
[96] Cardona R, Miranda E, Peralta-Salas D and Presas F 2023 Universality of Euler flows and flexibility of Reeb embeddings *Adv. Math.* **428** 109142
[97] Cardona R, Miranda E and Peralta-Salas D 2023 Computability and beltrami fields in euclidean space *J. Math. Pures Appl.* **169** 50–81
[98] Dyhr S, González-Prieto Angel, Miranda E and Peralta-Salas D 2025 Turing complete navier-stokes steady states via cosymplectic geometry (arXiv:2507.07696)
[99] González-Prieto Angel, Miranda E and Peralta-Salas D 2025 Universality in computable dynamical systems: old and new *J. Phys. Complex.* **6** 035014
[100] Cardona R, Miranda E and Peralta-Salas D 2025 Towards a fluid computer *Foundations of Computational Mathematics* pp 1–17
[101] Bausch J, Cubitt T S and Watson J D 2021 Uncomputability of phase diagrams *Nat. Commun.* **12** 452
[102] Gonda T, Reinhart T, Stengele S and De les Coves G 2024 A framework for universality in physics, computer science and beyond *Compositionality* **6** 3
[103] Purcell J, Zhi Li and Cubitt T 2024 Chaitin phase transition (arXiv:2410.02600)
[104] Siegelmann H T 1996 The simple dynamics of super turing theories *Theor. Comput. Sci.* **168** 461–72
[105] Pour-El M B and Richards J I 2017 *Computability in Analysis and Physics* vol 1 (Cambridge University Press)
[106] Copeland B J and Shagrir O 2018 The Church-Turing thesis: logical limit or breachable barrier? *Commun. ACM* **62** 66–74
[107] Kolchinsky A and Wolpert D H 2021 Work, entropy production and thermodynamics of information under protocol constraints *Phys. Rev. X* **11** 041024
[108] Hartle H, Wolpert D, Stier A, Kempes C P and Manzano G2024 Work extraction with feedback control using limited resources (arXiv:2407.05507)
[109] Kolchinsky A and Wolpert D H 2018 Semantic information, autonomous agency and non-equilibrium statistical physics *Interface Focus* **8** 20180041
[110] Wolpert D H and Kolchinsky A 2016 Observers as systems that acquire information to stay out of equilibrium *The Physics of the Observer Conf.*
[111] Karp R M 1992 On-line algorithms versus off-line algorithms: how much *Algorithms, Software, Architecture: Information Processing* vol 92 p 416
[112] Kosse C and Burdakov D 2019 Natural hypothalamic circuit dynamics underlying object memorization *Nat. Commun.* **10** 2505
[113] Bornschein J, Henniges M and Lücke J 2013 Are v1 simple cells optimized for visual occlusions? a comparative study *PLoS Comput. Biol.* **9** e1003062
[114] Li J, Leifer A M and Wolpert D H 2025 Measuring amount of computation done by c. elegans using whole brain neural activity (arXiv:2504.10300)
[115] Mota F, Aaronson S, Antunes L and Souto A 2013 Sophistication as randomness deficiency *Int. Workshop on Descriptional Complexity of Formal Systems* (Springer) pp 172–81
[116] Downey R and Hirschfeldt D R 2019 Computability and randomness *Not. Am. Math. Soc.* **66** 1001–12

[117] Ziv J and Lempel A 1978 Compression of individual sequences via variable-rate coding *IEEE Trans. Inf. Theory* **24** 530–6
[118] Zenil H, Kiani N A and Tegnér J 2016 Methods of information theory and algorithmic complexity for network biology *Sem. Cell Dev. Biol.* **51** 32–43
[119] Estevez-Rams E, Lora-Serrano R, Nunes C A J and Aragón-Fernández B 2015 Lempel-Ziv complexity analysis of one dimensional cellular automata *Chaos* **25** 123106
[120] Haslinger R, Klinkner K L and Shalizi C R 2010 The computational structure of spike trains *Neural Comput.* **22** 121–57
[121] Cover T and Thomas R 2006 *Elements of Information Theory* (Wiley)
[122] Li J, Leifer A M and Wolpert D H 2025 Measuring amount of computation done by c.elegans using whole brain neural activity (arXiv:2504.10300)
[123] Takens F 2006 Detecting strange attractors in turbulence *Dynamical Systems and Turbulence, Warwick 1980: Proc. Symp. Held at the University of Warwick 1979/80* (Springer) pp 366–81
[124] Roscoe A W 2010 *Understanding Concurrent Systems* (Springer)
[125] Reisig W 2016 Petri nets *Modeling in Systems Biology: The Petri Net Approach* (Springer) pp 37–56