Consciousness qua Mortal Computation

Johannes Kleiner[1,2,3,4]

[1]Munich Center for Mathematical Philosophy, Ludwig-Maximilians-Universität München
[2]Graduate School of Systemic Neurosciences, Ludwig-Maximilians-Universität München
[3]Institute for Psychology, University of Bamberg
[4]Association for Mathematical Consciousness Science

ABSTRACT. Computational functionalism posits that consciousness is a computation. Here we show, perhaps surprisingly, that it cannot be a Turing computation. Rather, computational functionalism implies that consciousness is a novel type of computation that has recently been proposed by Geoffrey Hinton, called mortal computation.

1. INTRODUCTION

A fundamental tenet of general purpose digital computing is that software is separated from hardware, so that the same program or algorithm can be run on any suitable system. This tenet is about to be broken. Contemporary developments in Artificial Intelligence (AI) and AI chip production have led to the identification of a novel concept of general purpose computing, called *mortal computation* (Hinton, 2022). This concept draws a line between the type of computations that contemporary processing units do, and the type of computations that brains and other biological organisms carry out.

Computational functionalism, first defined by Putnam (1967), posits, in a nutshell, that consciousness is a computation. This view has gained popularity again in light of the staggering achievements in AI development in recent years. AI models are computations, so if computational functionalism is true, AI models can—and, depending on the nature of the computation that consciousness is, will—become conscious (Butlin et al., 2023).

Here we show that computational functionalism is not indifferent with respect to the type of computation that consciousness is. We show that if there is any organism that is capable of conscious experiences, but which cannot be programmed—for example, non-human animals; cf. Assumption 1—, then computational functionalism implies that consciousness is a mortal computation. To establish this result, we make use of a differential definition of mortal computation, as well as general facts about the relation between programs, Turing computation and immortal computation.

Our result challenges the usual understanding of computational functionalism, which is centered around Turing-like models of computation. If our result holds true, consciousness cannot, according to computational functionalism, be a Turing computation or programmed. Yet, contemporary AI systems and programs are Turing computations. Therefore, this result speaks against the possibility of AI consciousness (though it does not aim to settle the issue due to questions of realization, cf. Section 7).

The underlying perspective of this paper is that the discovery of mortal computation by Hinton (2022) may well be a first step towards understanding of a whole new paradigm of computation, potentially as consequential as the Turing-Church-Gödel-Herbrand paradigm of computation of the past nine decades (Gödel, 1934; Church, 1936; Turing, 1937b).[1]

2. MORTAL COMPUTATION

The notion of mortal computation was identified and coined by Hinton (2022, Sec. 9),

who describes a learning task that makes use of unknown properties of hardware that vary across systems, such as variations in the connectivity of a system, or variations in non-linear processes in a system. As a result, the parameter values that define the learned computation "are only useful for that specific hardware instance, so the computation they perform is mortal: it dies with the hardware" (Hinton, 2022, p. 13). The general computing paradigm of the past nine decades, in contrast, implies that a computation is largely independent of the hardware on which it is run: "[T]he same program or the same set of weights can be run on a different physical copy of the hardware. This makes the knowledge contained in the program or the weights immortal: The knowledge does not die when the hardware dies" (Hinton, 2022, p. 13).

There is, at this early stage, no constructive definition of mortal computation,[2] but we may consider a differential definition, that helps us distinguish mortal computations in virtue of what they are not. To provide such definition, denote by $\mathcal{C}$ the class of all computations. $\mathcal{C}$ comprises all Turing computations, which we will denote by $\mathcal{C}_{\mathrm{TM}}$ in what follows, as well as other notions of computation, for example, non-deterministic Turing computations, neural computations, analogue computations and the yet-to-be-understood mortal computations.

The core intuition behind immortal computation is "that the software should be separable from the hardware" (Hinton, 2022, p. 13). In practice—in central processing units (CPUs), graphics processing units (GPUs), tensor processing units (TPUs), or data processing units (DPUs)—this separation is enabled by a processing unit's *Instruction Set Architecture* (ISA). An ISA contains specifications of all computations that the processing unit can carry out, and it is with respect to these specifications that programs, operating systems and compilers are defined. To run a program is to run machine code that specifies which of the ISA's computations are to be carried out in which order (call this concatenation) and how the results of computations are to be used by other computations (call this combination). Differences among processing units' performance, design, size, etc., are

differences in an ISA's *implementation*. The ISA exists to ensure binary-code compatibility of software despite such differences; it is the boundary between software and hardware.

The computations defined by an ISA constitute a reference relative to which software is defined, and which a class of hardware implements. It ensures that a program can run on different physical copies of the same type of hardware. Computation is immortal precisely because it is defined with respect to such reference. We can formalize this requirement as follows.

Definition 1. A computation $c \in \mathcal{C}$ is *immortal* iff there is a class of reference computations $c_{\mathrm{ref}} \subset \mathcal{C}$ such that c is a concatenation and combination of these reference computations. A computation c is *mortal* iff it is not immortal.

We will denote the class of immortal computations by $\mathcal{C}_{\mathrm{Imm}}$. Immortal computations are meant to be a subclass of Turing computations, so that we have

$$\mathcal{C}_{\mathrm{Imm}} \subset \mathcal{C}_{\mathrm{TM}} . \qquad (2.1)$$

Because an immortal computation c is a concatenation and combination of reference computations, every system that can realize an immortal computation c must be able to realize its reference computations c_{ref}. This is the only implication of Definition 1 we will make use of in what follows. To explicate this implication formally, we denote by *Sys* the class of all systems. This class includes, for example, all CPUs, GPUs, TPUs, and DPUs in use today, as well as all biological organisms. Furthermore, we denote by $\mathcal{C}(S)$ all computations that a system $S \in \mathit{Sys}$ can realize or implement. Using such formalism is of advantage because it can be applied to any account of implementation of a computation (Piccinini, 2015). The essential implication of the previous definition then reads as follows.

Implication 1. If $c \in \mathcal{C}$ is immortal, then there is a class $c_{\mathrm{ref}} \subset \mathcal{C}$ such that for all $S \in \mathit{Sys}$

$$c \in \mathcal{C}(S) \Rightarrow c_{\mathrm{ref}} \subset \mathcal{C}(S) . \qquad (2.2)$$

An important class of immortal computations are computations specified by writing a

program in some programming language; computations that are coded, that is, and compiled to run on CPUs, GPUs, TPUs or DPUs. We will simply refer to these computations as 'programs'. Programs are immortal because they are defined with respect to some programming language that in turn is defined, via its compiler, with respect to one or more ISAs.

We denote the class of computations that can be coded with any of the existing programming languages by $\mathcal{P}rog$. Because programs are immortal, we have

$$\mathcal{P}rog \subset \mathcal{C}_{\mathrm{Imm}} . \qquad (2.3)$$

3. Computational Functionalism

Computational functionalism was introduced by Putnam (1967) as the following set of assumptions.

1. "All organisms capable of feeling pain are Probabilistic Automata.

2. Every organism capable of feeling pain possesses at least one Description of a certain kind (i.e., being capable of feeling pain *is* possessing an appropriate kind of Functional Organization).

3. No organism capable of feeling pain possesses a decomposition into parts which separately possess Descriptions of the kind referred to in 2.

4. For every Description of the kind referred to in 2, there exists a subset of the sensory inputs such that an organism with that Description is in pain when and only when some of its sensory inputs are in that subset." (Putnam, 1967, 1975, p. 434)

In giving this definition, Putnam equates Probabilistic Automata with *descriptions* of a system; "[t]he Machine Table mentioned in the Description will then be called the Functional Organization of [a system] S relative to that Description" (ibid.).

The understanding of computation has evolved substantially since Putnam (1967), cf. e.g. (Piccinini, 2015). To connect Putnam's definition to computation as presently understood, and to do justice of it being a definition of computational functionalism, we reformulate Condition 2 in abstract terms, making use of the set $\mathcal{C}(S)$ of computations that a system can realize, which we have introduced above. It is clear from the context of Putnam's definition that it is to apply to all systems, not just organisms in a narrow sense. Denoting the experience of "feeling pain" by e, and the class of systems capable of having this experience by Sys_e, we may hence read Putnam's Condition 2 as follows.

Implication 2. Computational functionalism implies that there is at least one computation $c^* \in \mathcal{C}$ such that, for all $S \in Sys$,

$$S \in Sys_e \;\Rightarrow\; c^* \in \mathcal{C}(S) .$$

In Putnam's terms, being capable of realizing c^* is being capable of experiencing pain. Modulo details of sensory input referred to in Putnam's Condition 4, we may say that experiencing pain *is* realizing a computation c^*, or, in more simple terms yet, that the experience of pain is c^*. Nothing hinges on these terminological shortcuts, though. Putnam's conditions must hold as well for experiences other than pain, but may be realized by different computations in each case.

4. Programs

We have denoted the class of computations that can be coded with any existing programming language by $\mathcal{P}rog$. We now define Sys_0 to denote the class of systems that can run such programs. Because programs are defined relative to Instruction Set Architectures (ISA) of the underlying programming language, Sys_0 is the class of systems that can realize ISAs of existing programming languages. Denoting, as above, an ISA of a program $c \in \mathcal{P}rog$ by c_{ref}, this class is defined as

$$Sys_0 = \{S \in Sys \mid c_{\mathrm{ref}} \subset \mathcal{C}(S) \text{ for at}$$
$$\text{least one } c \in \mathcal{P}rog\} .$$
$$(4.1)$$

The class Sys_0 comprises all desktop and laptop computers, mobile devices, workstations, servers and supercomputers. It comprises anything that can run any Instruction Set Architecture of any existing compiler or programming language. But it does not comprise animals, or other organisms, that cannot be programmed—it does not contain organisms which are incapable of operating non-trivial logic as required by ISAs, that is. If

4

any such animal or orgamism is conscious, the following assumption holds true.

Assumption 1. There is a system $S \notin Sys_0$ that is capable of conscious experience e.

In what follows, we assume computational functionalism (viz. Implication 2) and Assumption 1. The following lemma shows that if this is the case, the computation c^* is not among all programs.

Lemma 1. $c^* \notin \mathcal{P}rog$.

Proof. Assume $c^* \in \mathcal{P}rog$ and let $\tilde{S}$ denote the system in Assumption 1. Because $\tilde{S} \notin Sys_0$, it follows that $c^*_{\text{ref}} \not\subset \mathcal{C}(\tilde{S})$. But because $\tilde{S} \in Sys_e$, Implication 2 implies that $c^* \in \mathcal{C}(\tilde{S})$. This violates (2.2), so that c^* cannot be immortal. But all programs are immortal (cf. (2.3)). Hence we have arrived at a contradiction. It follows that $c^* \notin \mathcal{P}rog$. $\square$

5. Turing Computation

Next, we consider Turing computations. A computation is a Turing computation iff it can be realized by (the abstract mathematical model of) a Turing machine.

Some (in fact, most) contemporary programming languages are Turing complete: they can be used to simulate universal Turing machines, meaning that they can be used to implement any Turing computation. For any Turing computation, one can write at least one program that realizes this computation, and running this program instantiates the Turing computation. This implies that

$$\mathcal{C}_{\text{TM}} \subset \mathcal{P}rog. \tag{5.1}$$

As a consequence, we have the following lemma, which shows that the computation c^* put forward by Computational Functioanlism is not a Turing computation.

Lemma 2. $c^* \notin \mathcal{C}_{TM}$.

Proof. Follows from Lemma 1 and (5.1). $\square$

6. Immortal Computation

The next lemma shows that consciousness is a mortal computation.

Lemma 3. $c^* \notin \mathcal{C}_{Imm}$.

Proof. Lemma 2 states that $c^* \notin \mathcal{C}_{\text{TM}}$. Because of (2.1), we furthermore have $\mathcal{C}_{\text{Imm}} \subset \mathcal{C}_{\text{TM}}$. Therefore, it follows that $c^* \notin \mathcal{C}_{\text{Imm}}$. $\square$

7. Conclusion

We have shown that computational functionalism implies that consciousness is a mortal computation, and that consciousness cannot be a program or Turing computation. We hope that this result contributes to the understanding of computational functionalism and its implications, including questions of AI consciousness, and that it highlights mortal computation as a potential concept of interest with respect to question of the mind.

Because all contemporary Artificial Intelligence (AI) is immortal computation, the results provide initial reason to believe that no current or near-future AI can be conscious. Only artificial systems that employ mortal computations can instantiate consciousness. The results presented here do not, however, prove this to be the case. That is because it might be the case that consciousness, despite being a mortal computation, can be realized by immortal computations. Whether this is a viable option, and what precisely it means to realize or implement a mortal computation, depends on details of the notion of mortal computation that are to be developed in future research. Because mortal computation is not Turing computation, the possibility of such realization might bear various difficulties, in case of which strong implications for synthetic phenomenology would follow.

Notes

[1]The question of whether a computation is a Turing *computation* is different from questions regarding Turing *computability*. The former concerns the nature of computations. For example, the question of whether neural computations are Turing computations (Piccinini, 2020). The latter concerns functions, in the mathematical sense of the term, that map natural numbers to natural numbers, and asks whether their value can be computed by a Turing machine. A function is Turing-computable iff there is a Turing computation (meaning: an abstract mathematical model of a Turing machine) that halts on all numbers for which the function is defined, and does not halt when provided with numbers for which the function is not defined. This is the case iff the function is λ-computable (Church, 1936; Turing, 1937a) or general recursive (Gödel, 1934; Kleene, 1936). The definition of Turing-computability of functions leaves open what the computation is that implements the function, which is what this paper is concerned with.

[2] There are two ways of reading Hinton (2022, Sec. 9). On a deflationary reading, a mortal computation is simply a Turing computation that is not known in its entirety to an outside programmer. Call this the epistemic reading of mortal computation. It is suggested by Hinton's emphasis of "large and unknown variations in the connectivity" (ibid.). On a different reading, a mortal computation is a computation that fundamentally transcends some of the constraints of Turing computation, for example the existence of an immutable tape for purposes other than read and write actions, or the existence of a transition function that is Markov, as suggested by Hinton's emphasis on "non-linearities of different instances of hardware" (ibid.). Call this the ontic reading of mortal computation. On the ontic reading, the state of affairs of the hardware is partially unknown to the computation itself. The computation may have to deal with, and make use of, non-Turing properties of the hardware. Both interpretations are compatible with (2.1).

References

Butlin, P., Long, R., Elmoznino, E., Bengio, Y., Birch, J., Constant, A., ... others (2023). Consciousness in artificial intelligence: Insights from the science of consciousness. *arXiv preprint arXiv:2308.08708*.

Church, A. (1936). An unsolvable problem of elementary number theory. *American Journal of Mathematics*, *58*(2), 345–363. doi: https://doi.org/10.2307/2371045

Gödel, K. (1934). On undecidable propositions of formal mathematical systems. In *Collected works* (Vol. 1). Oxford University Press, 1986.

Hinton, G. (2022). The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*.

Kleene, S. C. (1936). λ-definability and recursiveness. *Duke Mathematical Journal*, *2*(2), 340 – 353. doi: https://doi.org/10.1215/S0012-7094-36-00227-2

Piccinini, G. (2015). *Physical computation: A mechanistic account.* OUP Oxford.

Piccinini, G. (2020). *Neurocognitive Mechanisms: Explaining Biological Cognition.* Oxford University Press. doi: https://doi.org/10.1093/oso/9780198866282.001.0001

Putnam, H. (1967). Psychological predicates. In W. H. Capitan & D. D. Merrill (Eds.), *Art, mind, and religion.* Pittsburgh: University of Pittsburgh Press. (Reprinted in (Putnam, 1975).)

Putnam, H. (1975). The nature of mental states. In *Mind, language, and reality: Philosophical papers* (Vol. ii). Cambridge: Cambridge University Press.

Turing, A. M. (1937a). Computability and λ-definability. *Journal of Symbolic Logic*, *2*(4), 153–163. doi: https://doi.org/10.2307/2268280

Turing, A. M. (1937b). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, *s2-42*(1), 230-265. doi: https://doi.org/10.1112/plms/s2-42.1.230