Implementation-as: From Art & Science to Computing [Draft]

Nick Wiggershaus

Abstract. This paper vindicates interpretational accounts of physical computation. Specifically, recent agential approaches that couch implementation in terms of scientific representation are corroborated. Such accounts are strengthened by the introduction of a novel notion: *Implementation-as*. Implementation-as is theoretically underpinned by the DEKI-account (Frigg&Nguyen 2018), a formalized account of scientific representation relying on Goodman's and Elgin's notion of representation-as. The ensuing result is a philosophically robust account, satisfying the most important desiderata for accounts of computation in physical systems. The upshot is that physical computation occurs when agents *use* material systems as epistemic tools to compute a function. Application of this new framework is illustrated for the case of the MONIAC (an analog device) and the IAS-machine (a digital computer).

Keywords: Physical Computation; Implementation; Scientific Representation; Representation-as; MONIAC; IAS-machine

1 Introduction

Computability theory allows us to formally engage with computation in mathematical terms. Studying computation merely in the abstract though, does not provide any details about its physical implementation. The basic problems that any account of physical computation must answer are (Sprevak 2018; Ritchie & Piccinini 2018):

> **COMP** The conditions under which a physical system is computing.
> **IDENT** The conditions that specify that a computational system implements one computation rather than another.

Solving this so-called *Problem of Implementation* is important for disciplines such as the foundation of computer science, AI, robotics, and cognitive science. Subsequently, a huge literature of potential candidate frameworks has been presented. Virtually all accounts propagate that there is an equivalence relation between the computational formalisms of the mathematical theory of computation with the putative computing system. Formally, the idea is to establish a mapping between the sequence of states of an abstract model of computation and the state transitions of a physical system.

However, mappings are (too) cheap to come by. Putnam's (1988) and Searle's (1992) characterizations of physical computation for instance, merely call for a physical state to computational state correspondence. While simple and straightforward, such *simple mapping accounts* (SMA)[1] render both COMP and IDENT trivial. Accordingly, every (sufficiently complex) macroscopic object simultaneously computes all kinds of functions. As a reaction to the threat of pancomputationalism, a plethora of constraints were added to the SMA, resulting in what may be called *extended mapping accounts* (EMA):

[1] Baptized by Godfrey-Smith (2009) and arguably popularized by Piccinini (2015), such types of accounts are commonly called simple mapping accounts. I follow suit with that terminology.

Extended Mapping Account (EMA)

1. There is a function f mapping the states s_j of S_C to states of M_C, such that
2. Under f, the physical evolution/state transitions $s_j \rightarrow s_{j+1}$ are morphic to the formal state transitions $m_i \rightarrow m_{i+1}$ of M_C (specified by δ), where $f(s_j) = m_i$.
3. f is constrained by extra conditions.

Many philosophers concerned with computation argued that the material implication of physical state transitions is too permissive, allowing for too many computational structures. In due course, many argued that physical computation needs to obey *counterfactual* state transitions (see e.g., Copeland (1996)). In other words, if the system S_C had been in a physical state that maps onto m_i, it would have evolved into a state that maps onto m_{i+1}. Others formulated similar requirements in terms of a suited *causal* structure (Chalmers (1996), Scheutz (1999)) or *dispositional theories* (Klein 2008). Additionally, widely embraced refinements of the SMA have been formulated in mechanistic terms (e.g., Milkowski (2013), Fresco (2014), Piccinini (2007, 2015)). In so far as mechanisms have a causal structure/are said to have counterfactual state transitions, the mechanistic account can be interpreted to follow the strategy of constraining the SMA. Accordingly, computation must be implemented in specific computational mechanisms.

In order to judge such competing accounts of computation, Piccinini (2007; 2015) presented a convenient heuristic to evaluate them. Five desiderata were advanced:[2]

Desiderata of Physical Computation

(1) *Objectivity*: An account of physical computation should make it, at least in part, a matter of fact whether a system is implementing a computational function. The intention is to align computation with scientific practice and scientific objectivity.
(2) *Extensional Adequacy*: An account of computation should avoid triviality (the main shortcoming of the SMA); in slogan form, it should proclaim that the *right things compute* (laptops and perhaps brains) and the *wrong things do not compute*.
(3) *Explanation*: The computations performed by a material system should, at least partly, explain its behavior and capacities
(4) *Miscomputation*: Sometimes, computation goes wrong. An account of physical computation should account for faulty behavior.
(5) *Taxonomy*: An account of computation should be able to untangle the different computational capacities of different systems (e.g., general purpose or fixed purpose; analog, digital, or quantum).

To date, arguably no account of physical computation has championed all the others.[3] In fact, recent scholarship has seen the emergence of yet another new way to characterize concrete computation. The common denominator of this cluster of literature is to couch the implementation relation between M_C and S_C in terms of scientific representation and modeling. Importantly, these views do *not* suggest that computation comes about when an abstract computational model describes or represents a physical system – instead, it is the other way around! Simply put, the idea

[2] I follow a slightly adjusted version of Duwell (2021) which merged "the right things compute" and "the wrong things don't compute" under 'extensional adequacy'.
[3] Of course, it is a viable option to take a pluralistic stance with respect to accounts of computation.

is that concrete computing systems implement their 'target' (a model of computation) analogous to how material models represent their target. For instance, when developing a model of computation called *L-machines,* Ladymen (2009) suggests that physical computation might be contingent on (scientific) representation. Another case in point is Care's (2010) historical study shedding light on the use-centric history of analog computing as modeling. Likewise, but from a philosophical angle, Papayannopoulos (2020) highlighted the conceptual commonalities between analog computers and analog models (when developing a notion of analog computation). Arguably the technically most detailed account in that vein today is the *Abstraction/Representation (AR) Theory* introduced by Horsman, Stepney, Wagner, and Kendon (2014) and developed further in several publications.[4] Horsman and collaborators provide sophisticated 'commuting diagrams' in virtue of the *representational triple* $\langle m_i, f, s_j \rangle$, where *f* is perceived as scientific representation, and m_i and s_j corresponding computational and physical states, respectively. Subsequently, Fletcher (2018), Szangolies (2020), and Duwell (2021) critically assessed (AR) Theory under philosophical considerations and concluded that the approach is a viable contender when formulated in agential terms. And more recently Wiggershaus (2023) argued that such agential accounts offer an avenue to unify different notions of implementation in computer science.

I believe that the development to philosophically characterize computation in virtue of scientific representation is worthwhile pursuing because it offers acumens of an already established discourse on how to render an equivalence relation between mathematical entities and physical objects. However, so far, there remains a lacuna with such views: The range of accounts of scientific representation has mushroomed in recent decades and is at least as nuanced as the field of physical computation. Accordingly, we need to answer 'Which account of scientific representation should one rely on?' – else this novel approach remains uninformative or even disputable.[5] This paper sets out to respond to such issues by developing the novel notion of *implementation-as*. Implementation as specifically builds upon the DEKI account of scientific representation (which was devised by the notion of *representation-as* from the philosophy of art).

In what follows, the paper is organized as follows: Section 2 introduces the notion of representation-as by Elgin and Goodman, based on which the DEKI account was devised. To facilitate discussion, I follow Frigg and Nguyen in introducing their account in terms of the MONIAC (a hydraulic analog computer). Subsequently, in section 3, I transpose the features of the DEKI account to the realm of computing. Thereafter, section 4 demonstrates how implementation-as applies to a proper digital computing device – the historical example of the very influential IAS-machine. Finally, before concluding, I briefly evaluate the here newly introduced agential notion of implementation along the five desiderata of physical computation.

[4] Horsman (2015, 2017), Horsman Kendon, Stepney, (2017, 2018) and Horsman et al. (2017).

[5] Utilizing different notions of scientific representation may result in significantly different accounts of concrete computation. For instance, if one were to subscribe to a view like Suppes' (2002), i.e., scientific representation is a relation merely reducing to isomorphisms between structures, then computation would be too (nowadays, the 'isomorphism-view' is considered questionable though Suárez (2003)). If, on the other hand, one were to follow Cohen & Callender (2006), according to whom anything may represent anything else, then scientific representation-based computation would be in danger of collapsing into pancomputationalism.

2. Scientific Representation, Representation-as, & DEKI

2.1 From Art to Science

Scientific representations concern a wide array of phenomena. One may use diagrams, mathematical equations, or material objects for representations in science. Most generally, any representation that is the result of scientific practice may be deemed a scientific representation. In this paper, we are primarily interested in the case of (material) scientific models and how they represent.

According to the *representational conception* by Giere (1999), scientific models are used by scientists to represent some (real-world) system. Scientists use models and their representational capacities as a *surrogate* to *reason* about target systems (e.g., explanation, prediction, confirmation) (Swoyer 1991). Scientific representation then may be characterized as the relation *f* between a model *M* and its dedicated target system *T*, such that $f: T \rightarrow M$. One may use material or theoretical models and either concrete or hypothetical target systems (Weisberg 2013). Despite this seemingly simple conception, philosophers of science identified many problems and questions associated with scientific representation in recent decades (Frigg&Nguyen 2020a).

One successful problem-solving strategy has been to seek answers in the study of art and languages. A case in point is the notion of representation-as, introduced by Nelson Goodman and Catherine Elgin (Goodman 1976; Elgin 1983). According to their theory of symbols, there are three fundamental 'modes of reference': (i) representation-of; (ii) Z-representation; and (iii) representation-as. This tripartite distinction stems from the observation that many representations represent an object as something else. A common pictorial example is caricatures. Take for instance the depiction of Winston Churchill as a bulldog. Letting '*X*' stand for the representing thing (a caricature); '*Y*' for the thing represented (Winston Churchill); '*Z*' stands for the kind of representation (a bulldog). The caricature features all the relevant distinctions of representation at once. First, the caricature is a representation-of Churchill, because it denotes the former English Prime minister. Secondly, the caricature is also a *Z*-representation, where here '*Z*=bulldog' since it exemplifies the features of a bulldog. Thirdly, the caricature represents Churchill as a bulldog, because the bulldog features (such as being stubborn or resilient) are imputed to him. In the remainder of the paper, such XYZ-triplets with their corresponding notions of denotation, exemplification and imputation will be chief for understanding the notions of representation-as and implementation-as, respectively.

Subsequently, philosophers such as Hughes (1997), Elgin (2010, 2017), and van Fraassen (2008) appropriated the representation-as conception to the scientific realm. In what follows, I introduce what arguably is the most sophisticated of such accounts: Frigg and Nguyen's DEKI account.

2.2 The DEKI account

In a recent number of publications, Frigg & Nguyen (2017 2018, 2020a, 2020b) introduced their so-called DEKI account, providing a full-fledged and systematized account of scientific representation based on representation-as. The DEKI account applies both to material models and non-concrete models. I follow suit with the authors to discuss the account based on a material

model – the Philips-Newlyn machine (also known as *MONIAC*).[6] As we will see, under different assumptions, the very same device may be regarded as a special purpose hydraulic analog *computer* instead of a scientific model. It thus serves as an ideal gateway for establishing a link between scientific representation and implementation.

Before introducing the DEKI account's most salient features, I first acquaint ourselves with the MONIAC to have a more demonstrative discussion.[7] Standing about 2m tall, more than 1m wide and almost 1m deep, the device comprises several see-through plastic tanks and tubes filled with colored water. Attached to the tanks are pulleys, sluices, gauges, and pens (used to plot graphs). The design of the machine uses pumps and gravity to let water accumulate in different reservoirs containing floats that drive the different components in the mechanism depending on the water level. *Qua* scientific model, the purpose of the machine is to model a national economy by the circular flow of water – the flow of the water stands for the exchange of commodities. Each of the machine's tanks corresponds to different features of an economy (national income, governmental spending, etc.). Depending on the configuration of the mechanical components of the MONIAC, different amounts of water accumulate in the different tanks, allowing to model various economic scenarios. Fig. 1 shows a simplified scheme of these components and how they enable the device to work in connection with the notion of representation-as.

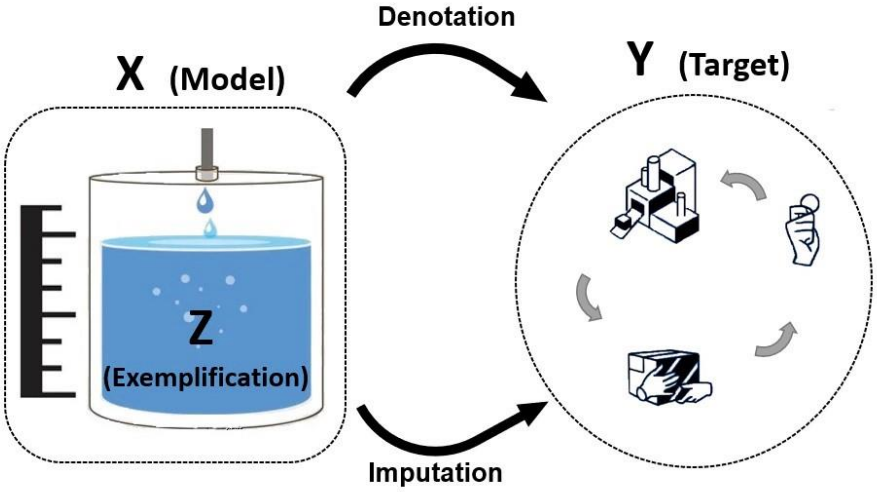

Fig. 1: Schematic depiction of the MONIAC at work, representing an economy through the flow of water. Applying the XYZ-triplet and the corresponding notions of denotation, exemplification, and imputation to a scientific context results in the notion of representation-as. Roughly put, *X* takes on the role of the (material) model (e.g., a tank filled with water); *Y* takes on the role of the target (e.g., an economy); *Z* takes on the role of the exemplified features of the representing object.

[6] The name MONIAC (standing for 'Monetary National Income Analog Computer') is more common in the US, where the coinage of the term was due to economist Abba Lerner "to suggest money, the ENIAC, and something mechanical." (Fortune 1952, 101).

[7] Multiple authors have provided technical descriptions of the machine, its underlying economic theory, and its history (see e.g., Phillips 1950; Newlyn 1950; Barr 1988; Bissel 2007; Morgan 2012, 172-216).

However, there is an important difference when applying the XYZ-triplet to models like the MONIAC as opposed to caricatures. Whereas the latter can rather straightforwardly be identified as e.g., a bulldog-representation, it is much less obvious how the MONIACs water-filled pipes and tanks are supposedly an economy-representation. The problem is that the machine does not instantiate actual economic features. For the sake of modeling, scientists hence need to *translate* the flow of water into the 'flow' of commodities under an agreed-upon interpretation. As Morgan and Boumans (2004) explain, there is a long tradition in economics of relying on the *metaphor* that certain economic things behave like water. For such metaphors to be useful, scientists may only use the MONIAC as an economy-representation under a specific, highly non-trivial interpretation. "When we come to build a model based on the metaphor, we have to make commitments about exactly what we mean." (Morgan and Boumans 2004, 8). As I will explain further down below, similar commitments about specific interpretations are paramount for computation, too.

Frigg and Nguyen suggest formalizing these considerations through Elgin's and Goodman's analysis of representation in the art world. In case of the MONIAC,

> "[…] the idea behind the machine is that hydraulic concepts are made to correspond to economic concepts. This means that we turn system of pipes and reservoirs into an economy-representation by interpreting certain selected X-features as Z-features. The water in a certain reservoir is interpreted as money being saved; the level of water in the reservoir is interpreted as a quantity of money; and so on." (Frigg & Nguyen 2020a, 166)

Denotation, exemplification, and imputation thus constitute the core of representation-as and find application in their full-fledged account of scientific representation. To be informative in the scientific arena though, a fourth element – the notion of a 'key' – is introduced. Keys are meant to adjust model features to target features, because typically model features can rarely be transferred unaltered to a target (e.g., one may need a scale factor or a conversion of units). Together these four salient features form the acronym DEKI. In sum, the following picture emerges:

DEKI-account

A model is defined as an ordered pair $M = \langle X, I \rangle$, where X is an object and I is an interpretation. I is what turns a selected object X into a model. M represents Y as such and so *iff* conditions (1)-(4) are met:

(1) An interpreted object X (the model M), like the MONIAC, *denotes* a target Y (e.g., the British economy).

(2) M *exemplifies* Z-features. For instance, to be an economy representation, the MONIAC needs to exemplify economy-features (Z-features). However, often scientific models do not directly exemplify the required Z-features. The MONIAC e.g., is nothing but a sophisticated collection of pipes and tanks filled with water; it only has such-and-such dimensions, weighs so and so many kg, has n-number of components, etc. It merely instantiates the flow of water; it does not realize economic features such as the exchange of commodities. To turn such a model's features into the required Z-features, we need to resort to the interpretational capacities of the designers and users. Only under a specific agreed-upon interpretation I are the scientists licensed to translate features of their model into Z-features $I: X \rightarrow Z$.

(3) There is a *key K* that systematically *translates* the exemplified *Z*-features $\{Z_1,\ldots, Z_n\}$ of the model, into another set of *Y*-features (the features of the target). In the case of the MONIAC, units of volumes of water (that are interpreted as the flow of commodities) must be translated into units of a specific currency. Furthermore, the time of the machine operating must be translated into the time of economic cycles. Depending on the denoted target, a key may associate one liter of water with e.g., 1 million pounds or 5 million US dollars.

(4) *M imputes* at least one of the 'keyed-up' features to the target. If the users of the MONIAC are interested in say, only tax revenue, they might only impute one single feature (corresponding to tax revenue) to the target.

In sum, the result is an intentional conception of scientific representation, as all its features (1)-(4) require different interpretations in the form of intersubjective agreements of the scientists using them. Through the selection of an appropriate material system, target phenomena are represented as something else. The MONIAC for instance represents the flow of money as the flow of water.

3. From Science to Computing: Implementation-as

Transposing the just introduced DEKI framework to the notion of implementation in computer science results in the introduction of the novel notion of *implementation-as*. The successful transposition requires an adaptation of the original DEKI account to the computing context. In the following four subsections, I illustrate how the adjustment from the scientific arena to computing plays out. The discussion unfolds along the most salient features of the DEKI account, viz., denotation, exemplification, keying-up, and imputation.

3.1 Denotation

Generally, we need to think of denotation as the dyadic relation of a name (or label) and a bearer it applies to. The relation is established by an interpretive act. Elgin, for instance, states that "[r]epresentation- of— that is, denotation— can be achieved by fiat. We simply stipulate: let x represent y and x thereby becomes a representation of y.", (Elgin 2017, 253). Whilst originally a linguistic concept, she argues that there is nothing intrinsic in the notion of denotation that would restrict it to language only. Both symbols and what they denote can be of many different types. Consequently, Goodman and Elgin both apply denotation to other instances:

> "Pictures, equations, graphs, charts, and maps represent their subjects by denoting them. They are representations of the things that they denote. [...] It is in this sense that *scientific models* represent their target systems: they denote them." Elgin (2010, 2; own italics)

In the scientific context, denotation is taken to establish a connection between a model *X* and its intended target *Y*. Put differently, denotation establishes which target is supposed to be represented. Now, I submit that denotation applies *mutatis mutandis* to physical computation.

At first, this one may not strike as surprising for denotation is also not an unfamiliar notion in computing. For instance, the notion of denotational semantics is paramount for computer scientists to formally determine the meanings of programming languages. Likewise, when following popular interpretations that computers are symbol manipulators, one may subscribe to the view that the manipulated symbol structures denote information, data, etc. In the literature of physical computation, the so-called *semantic accounts* turn such a reading into a philosophical approach: as

Fodor (in)famously proclaimed, there is "no computation without representation.", (Fodor 1981, 180). The slogan especially embraces the metaphysical assumptions underpinning those branches of cognitive science that maintain that the brain computes. Exemplary of the 'aboutness' of neural computation is Marr's hypothetical case of the apocryphal grandmother cell (a cell that fires only when one's grandmother is in sight) (2010, 15). Today, semantic accounts may come in vastly varying degrees of commitment to what kind of processing of representations is essential for computation. More recent versions, for instance, may share the most salient constraints of some of the EMAs (e.g., causal, counterfactual, or disposition) but call for the additional condition that computational states must be representational (see Shagrir (2020) for an overview).

However, implementation-as should not be characterized as just another semantic account. Importantly, when it comes to implementation-as the choice of the potentially denoted target is restricted to the to-be-implemented sequence of computations. So, in contrast to Marr's example, denotation may not be used to establish a dyadic relation to one's grandmother or any other external events, etc. Here the notion is *exclusively* reserved for the relation between a material system and a computational formalism *P* which specifies a sequence of computations.

> **Denotation**: Establishing which computational formalism *P* is supposed to be implemented in the putative material computing system.

As such, one of the key features of denotation (as a stipulative act) is that it enables the programmers and users to *specify* which sequence of computations ought to be implemented. Without denotation, we were not able to determine which computational formalism or program *P* (instead of *Q*, *R, S*, …) is originally intended to be run by the material device. What's correct behavior in the execution of *P*, may count as malfunctioning (miscomputation) of *Q*. And without knowing what is supposed to be computed, we would be unable to judge correct implementations from faulty ones. A prominent case from the philosophical literature is captured by Kripke's remark about Wittgenstein's hypothetical *rule-following* machines:

> "How is it determined when a malfunction occurs? By reference to the program of the machine, as intended by its designer, not simply by reference to the machine itself. […] Whether a machine ever malfunctions and, if so, when, is not a property of the machine itself as physical object but is well defined only in terms of its program, as stipulated by its designer.", (Kripke 1982, 34f)

Assigning a physical system or device with performing a certain task rather than another is not exclusively limited to computation, but rather ubiquitous to technology. In computing specifically though, we then assign the teleological function to compute a specific mathematical/computational function *P* to a material system; denotation is chief for specifying which computational function *P* is supposed to be implemented. What makes function ascription (in the teleological sense) a special case when it comes to computing is that we exclusively assign the execution of a rule or *mathematical/computationa*l function to a system. When assigning teleological functions like brewing coffee to a coffee machine, driving screws into a wall to screwdrivers, etc., the assigned functions concern physical properties and activities (e.g., pouring hot water onto ground coffee) and not formal, mathematical, or computational ones. This raises the question, how can concrete material systems exemplify computation?

3.2 Exemplification

In principle, all different kinds of physical properties can be used as computational vehicles. Computing systems may work based on the change of mechanical components, electronic components, biological components, and so on. Accordingly, computations are commonly held to be *multiply realizable*. It is this multitude that raises the vexing issue of COMP – given that, almost anything can be used for implementation, how can one determine its extensional adequacy?[8] While previous accounts of physical computation came up with various ingenious conditions and constraints to the SMA to answer which systems compute (and which ones don't), the current contribution frames the problem in terms of exemplification.

As we have seen, different objects and systems exemplify different properties in different manners. Caricatures may exemplify bulldogs by pictorial means. And beyond the art world, properly working coffee machines, for instance, exemplify some concrete mechanism allowing them to brew coffee. In the scientific context, we established that models like the MONIAC additionally require an interpretative component, for the hydraulic device does not literally exemplify economic properties on its own.

Now, I maintain that physical systems exemplify computational properties in virtually the same way as interpreted models exemplify their features – through interpretational exemplification. Put differently, interpretational exemplification allows physical systems (the putative computing device) that would otherwise simply count as mechanical, hydraulic, or electronic, to be turned into computing systems.

> **Interpretational Exemplification**: I: $X \rightarrow Z_C$. Turning selected X-features into computational states Z_C through an interpretation.

Importantly, two components are necessary for interpretational exemplification to be fruitful. On the one hand, a suitable kind of interpretation. On the other hand, we also require a *suited* physical substrate (X-feature) as well. As I will explain below, not every arbitrary object is suited to be a computational vehicle. Only when these two combined elements act in concert, does this novel hybrid approach avoid the pitfalls of interpretational pancomputationalism.

To elaborate on how this works, we need to have a closer look at how 'interpretation' is employed. In the philosophical literature, interpretational accounts of computation have often been shunned for being overly flexible. Without any constraints on the interpretational freedom, potentially every material object could be trivially turned into a computer by mere stipulation. The trick to overcoming these worries is to remember that we already solved a similar issue when using the MONIAC and its flow of water as a metaphor to represent the exchange of goods in an economy. For instance, when presenting the DEKI account the authors remind us that

> "[w]hile one is initially free to choose [X]-properties and Z-properties freely, once a choice is made, representational content is constrained. […] Free choices, once made, are highly constraining. This is why models are epistemically useful." (Frigg & Nguyen 2018, 214)

In the same vein, it is a necessary condition that the selected X-features (and their interpretation as computational states) of the computing system need to be *held fixed*. In other words, the interpretation needs to turn selected features of the material device into computational states under a *one-to-one relation*. Choosing a different set of states $X = \{X_1,\ldots, X_n\}$ and turning them into

[8] I.e., what counts as proper computation and what doesn't; cf. (2) of the adequacy criteria.

computational states requires a new interpretational process for every new candidate set of computational vehicles. Whilst this fixation is necessary to employ objects like the MONIAC as a computational device, they do not suffice though. Taking a rock or wall, arbitrarily picking out some of their properties as *X*-features and holding these fixed, still does not turn them into useful computers

That's where the second crucial element of exemplification comes into play. Importantly, appropriate interpretations should only be applied to physical features or carriers that demonstrate a sufficient degree of *counterfactual state transitions*. This demand finds its way into the current account since it is both in line with the literature of scientific representation and an overwhelming consensus in physical computation discourse. Consequently, agents need to choose (and oftentimes build) potential computational vehicles that exhibit a reliable degree of counterfactual dependence. Such counterfactual support is chief for using both scientific models for surrogate reasoning and turning computational devices into epistemically fruitful instruments. Compare the following two quotes. Concerning scientific models, Bokulich for instance reminds us that

> "[…] in order for a model M to explain a given phenomenon P, we require that the counterfactual structure of M be isomorphic in the relevant respects to the counterfactual structure P. That is, the elements of the model can, in a very loose sense, be said to "reproduce" the relevant features of explanandum phenomenon." (Bokulich 2011, 39)

In the same vein, Piccinini provides a summary in his (2015, 19-25), showing that it is wide consensus that the microphysical state transitions of a material system that is deemed computing, requires counterfactual support:

> "In other words, the pure *counterfactual account* requires the mapping between computational and microphysical descriptions to be such that the counterfactual relations between the microphysical states are isomorphic to the counterfactual relations between the computational states." (Piccinini 2015, 19)

What this means in the case of the MONIAC is that different calibrations of the knobs, valves and tanks filled with water need to bring out reliable changes in behavior. 'If the input/initial conditions had been different' the output must be different accordingly. Such counterfactual support is crucial for the implementation of a computational function. Only if the *X*-features are chosen in such a way that different set-ups yield different interpretable outputs can material models/computers such as the MONIAC be used to model target systems like an economy or a computational formalism.[9] Controlling these counterfactual dependencies of computational devices is what enables to physically program these machines and use them to compute functions.

3.3 Encoding a Labeling Scheme

To recap, while denotation specified which computational formalism is supposed to be implemented, interpretational-exemplification imposes which properties of a putative computing system are taken to be as computational states. So far, these two steps are insufficient for the implementation of computations, for we only determined that something may act as a computer (not what it actually computes). Scholars of physical computation widely agree though that one

[9] These computational states correspond to a model of computation; in the case of the MONIAC, the model of computation is characterized by a set of differential equations. Often, the seminal paper by Pour-El (1974) is taken as the theoretical basis for models of analog computation. For a survey of such different models see Bournez & Pouley (2021).

needs to specify the conditions that a computational system implements one computation rather than another (IDENT). Now, in order to relate exemplified computational states to a specific model of computation, we need to define for what kind of computations they are employed.

One crucial aspect for determining such a computational profile is to allude to the notion of a *key*. According to DEKI, exemplified properties are 'keyed up' with properties that are supposed to be imputed to the target. While the name 'keying-up' is inherited from the DEKI account, I suggest resorting to the more common terminology used in computing, where the discussion is usually framed under the label of *encoding* or fixing a *labeling scheme* (cf. Copeland (1996)).

> **Encoding a labeling scheme**: Relating the set of interpreted computational vehicles Z_C with a set $P=\{P_1,\ldots, P_j\}$ of states that are presumed to be imputed to the targeted computational formalism.

In what follows, I introduce the arguably two most relevant types of encodings for computing.[10] The two types roughly correspond to analog and digital computers respectively.[11]

The first type of encoding essentially hinges on the same idea as the keys employed in material (scale) models. Certain physical magnitudes of the material model are selected (through exemplification) to scale with some chosen features of a target system. Weisberg (2013) for instance, discusses this at length based on the *San Francisco Bay–Delta model* and more recently Pincock (2022) based on a scale model of *Lituya Bay* for modeling giant rockslides generated impulse waves (a tsunami). Importantly, in the majority of cases, one cannot simply take the chosen *X*-features and directly impute them to the chosen target *Y*. In the case of the just mentioned scale models e.g., the key is not simply equivalent to the scale factor, because one must take into account that the fluid dynamics doesn't scale completely proportional.[12] Very similar keys are necessary for scaling in *analog computers* in general. Ulman, for instance, describes that machine units of a given analog machine must be adjusted to the denoted computational problem (cf. Ulman (2013, 55 and 123-14) and Ulman (2020, §2.1 and 58)).

Based on the work of Lewis (1971), Maley formalized this idea, developing the so-called *Maley-Lewis account* that's supposed to cover the case of analog computation. Simply put the Maley-Lewis account captures the idea of scaling, i.e., the more the representing physical magnitude Z_C increases or decreases (in a systematic way), the more the property that's denoted in- or decreases. These insights yield the formulation of the first type of encoding (cf. Maley (2011, 124)):

> **Type 1: Encoding (Scaling) by magnitude.** As Z increases (or decreases) by a margin *d*, Q increases as a *linear* function of X+*d* (or X-*d*); *E*:Z →*P*.

When it comes to the implementation of *digital computation* though, a digital labeling- scheme is needed. As Maley explains, numbers are typically represented by (i) a series of digits (numerals) and

[10] Whether the two types of keys are exhaustive or not, such that there might be other kinds of keys relevant for computing – for instance, in the case of quantum computing – is the subject of future research.

[11] In the context of computing, the digital/analog distinction is a vexed issue; simply put there are two major camps: According to one view, analog computation is understood as an *analogy* (the behavior of a damped spring-mass might be modeled by electronic components that analogously showcase similar behavior); according to the second view, the operation of an analog computer should be understood based on the manipulation of continuous values. An in-depth exorcism of the analog/digital distinction lies beyond the scope of this paper.

[12] And in the case of the MONIAC, we don't even have a scale model of a Keynesian economy at all, but an object where certain features are selected (X-features) such that their covariation tells us something about the denoted target Y. Remember, physical quantities like 'flow of water' must be related to 'flow of money' via a system of units.

(ii) a base.[13] A digit series is then interpreted as the relative value of the digits. Translating this idea into a digital version of a key, the second type of encoding is defined as:

> **Type 2: Encoding digitally (labeling scheme).** A digital encoding *E*: Z →*P* represents a number/symbol via its digits, where 'digit' means a numeral in a specific place. In addition, we require a base, which is used to interpret the relative value of digits.[14]

Having elucidated how to determine a computational profile, implementation-as requires a final step.

3.4 Imputation

Lastly, *imputation* is the final necessary component of the implementation-as framework. As a first stab, "imputation can be analyzed in terms of property ascription", (Frigg and Ngyuen 2018, 217). Let me briefly return to the scientific modeling context for the sake of clarifying what kind of properties are ascribed to what. When scientists use a scientific model to reason about a target system, they must be able to ascribe features of the former to the latter $f: T \rightarrow M$. Put differently, we may thus say that the model imputes features to the target. The MONIAC, a material model, imputes its exemplified (under an interpretation) economic features to the dedicated target. I propose to appropriate this practice to computing, such that *material* systems implement a computational formalism (the analog to the target) by relying on imputation.

The reason why we appropriate imputation from representation-as to computing is that we want to systematically relate the interpreted and encoded computational vehicles of a material system to the denoted computational formalism (cf. steps (1)-(3)). As such, imputation has a comparable function to the mathematical notion of a morphism (relating physical states and abstract computational states) evoked by the EMA.

> **Imputation**: Ascribing encoded computational states to a computational formalism.

But what are the ramifications of referring to the relation as an 'imputation' instead of a mapping? The philosophically relevant message is that the mapping is *stipulated by human agents*: As an agential theory of implementation, implementation-as relies on a, at least partly, mind-dependent notion of computation – we use devices as an aid for our computational goals which otherwise would need to be carried out by hand or in one's head. Imputation can be understood as the notion that relates the interpreted and encoded computational vehicles of the surrogate system we use for computation with the computational problem we wish to be solved. Implementation-as advocates for a stipulated implementation-relation. Such a relation has two principal advantages.

First, the advantage of a stipulated implementation relation is that it does not stand at odds with the state-of-the-art insights of applied mathematics. Called the *application-* or *bridging problem*, philosophers of applied mathematics seek to address the notorious issue of how the mathematical relates (or bridges) to the physical. In a nutshell, the problem is that mere morphisms between physical states and mathematical/states do not obtain, because strictly speaking functions only obtain between set-theoretic structures (and physical substrates do not offer such a unique

[13] By understanding 'numbers' in a loose sense, the method can be applied to symbols that are part of an alphabet.
[14] Formally, the digital representation of a number '$d_n\ d_{n-1} \ldots\ d_1\ d_0{}'$ is captured by the formula $(d_n \times b^n) + (d_{n-1} \times b^{n-1}) + \cdots + (d_1 \times b^1) + (d_0 \times b^0)$. In base 10 "sixty-five" e.g., is hence represented as "65" (6*10)+(5*1).

structure (Psillos 2006, van Fraassen 2008)). In response, most recently suggested solutions to the bridging problem state that the mappings between the physical and mathematical are mind-dependent (Pincock 2004, Batterman 2010, Bueno&Colyvan 2011, Nguyen&Frigg 2017). Put differently, at least some stipulations of agents are needed to *create* a structure and hence bridge the gap between abstract mathematical objects and concrete physical states.

Now, in so far as theories of implementation need to spell out how logico-mathematical models of computation relate to the physical, the problem of implementation is a special instance of the application/bridging problem (Wiggershaus 2023). Therefore, if not specified otherwise, accounts of physical computation should preferably be in line with the insights of the philosophy of applied mathematics. Imputation (a mind-dependent notion) is explicitly compatible with this demand. Accordingly, computational vehicles are associated with the logico-mathematical states of the implemented computational formalism.[15]

The second advantage and essential feature of imputation is that it bears a normative component – the pairing of exemplified features with features of the computational formalism can be right or wrong, hence explaining miscomputation. What's right is determined by the denoted computational formalism. Again, mere morphisms seem to fail the miscomputation-desideratum.[16] While the denotation-relation constitutes what is supposed to be implemented, imputation is the relation that pairs exemplified computational states and formal computational states (of the target). Only when imputation matches *all* the elements of the physical computational states required for a series of computations, then the denoted program P might be implemented correctly. Strictly speaking, if there is a mismatch, the system may compute in a way it should not; it is said to miscompute.[17]

3.5 Taking Stock

Subsuming the various elements appropriated from the scientific representation discourse results in a novel agential theory of implementation:

Implementation-as

Let the ordered pair $C=\langle X, I\rangle$ be a computational device, where X is a material system and I an interpretation. Let P be the computational formalism/program. C implements P as Z_C *iff* all the following conditions are satisfied:

(1) C denotes P.

(2) C exemplifies Z-properties $Z_1,\ldots,Z_n$ under and interpretation $I: X \rightarrow Z_C$.

(3) C comes with a computational encoding associating the set $\{Z_1,\ldots, Z_n\}$ with a (possibly identical) set of properties $\{P_1,\ldots, P_m\}$. $E\{Z_i\}=\{P_1\}$

(4) C imputes at least one of the properties $P_1,\ldots, P_m$ to P.

---

[15] The argument may pose a problem for naturalized or mind-independent theories of implementation. The seriousness of this threat may be subject to future research.

[16] For essentially the same argument against using morphisms in accounts of scientific representation see (Suárez 2003).

[17] There are various ways in which this computational norm can be broken. Fresco & Primiero (2013), offer a detailed taxonomy of the miscomputation of *software*, stating that miscomputation can occur at any level of abstraction, ranging from faulty specifications, through the algorithmic level, down to the machine. At the abstract physical interface, errors might be due to wear and tear or insufficient counterfactual support (Schweizer 2019, 38-40).

The resulting framework is baptized *implementation-as* for acknowledging the influence of the representation-as from the art and science. More explicitly, the account structurally resembles Frigg and Nguyen's DEKI account of scientific representation (of material models). For remember, instead of using a material scientific model (based on an interpreted object *X*) to represent a target system *Y* as thus-and-so, the core idea here is to use a physical system (based on an interpreted object *X*) to implement a series of computations or a program *P*. Put simply, the computational formalism can be regarded as the target that's supposed to be implemented. Both scientific representation and physical implementation are instances of object-based reasoning. In the former case, we manipulate and interpret a material model as a surrogate to reason about a target system/phenomenon. Concerning the latter, we configure and manipulate (i.e., program) a physical computing system to obtain the result of a computational function. As such, almost the entire DEKI-analysis of the MONIAC *qua* scientific model equally well applies to the machine when interpreted as an analog computer. Rather than representing a national economy, the device implements a specific model of computation.

Having spelled out the main features of implementation-as, the remainder of the paper demonstrates how implementation-as applies to a case study (sect. 4) before philosophically evaluating the novel theory (sect. 5).

4. Case Study: The IAS-machine

The goal of this section is to briefly illustrate how the notion of implementation-as can be insightful beyond theoretical discussion and the peculiar case of a special purpose analog hydraulic computer. For so doing, I apply the theoretical framework I have just outlined to a well-known and influential device: The *IAS-machine*.[18] It embodied the architectural principles of what's nowadays still commonly used and referred to as *von Neumann architecture*. I first introduce the device's basic components and how it was programmed in some detail (sect. 4.1), before demonstrating how implementation-as sheds light on how it implements physical computation (sect. 4.2).

4.1. Technicalities and Programming

The IAS-machine was one of the first binary stored-program computers, storing instructions and data in the same memory. For enabling these features, different components need to act as different computational states. The designers relied on vacuum tubes for the circuitry and Williams tubes (cathode ray tubes) for the memory. These components then formed three basic units: [19]

1. The main memory unit (M)
2. The Central Processing Unit (CPU): Containing Control-Unit (CU) and Arithmetic-Logic Unit (ALU)

[18]Multiple authors have provided technical descriptions of the machine, how it was programmed, and its history (Burcks et al. 1946; Estrin 1952; Ware 1953; Bigelow 1980; Burcks 1980; Aspray 1990; Priestley 2018). The device is a stored-program digital computer. Constructed over the course of six years by a team of scientists and engineers under the leadership of von Neumann at Princeton's Institute of Advanced Studies (IAS), the machine was finalized in 1952. The IAS-machine had a huge influence on future generations of computers in and outside of the industry, both in the US and overseas e.g., ILLIAC, MANIAC (in Los Alamos), and the IBM 701 (Aspray 1990, 86-94).

[19] These elements (or "main organs") were mentioned in different forms by von Neumann (1945), where they were called CA (central arithmetical), CC (central control), M (memory), I and O (input and output devices) and R (some external recording medium).

3. The Input/Output device (I/O)

Considering the functioning of these units and their underlying components in detail further clarifies our understanding of how exemplification and encoding work in the case of a stored program digital machine. So, let me briefly look at each of these units in detail, starting with the memory.

The memory was of 'Williams type' and composed of 40 standard commercial "off the shelf" (Bigelow 1980, 302) 5CP1A cathode ray tubes (relying on the emission properties of cathode-ray-tube phosphor screens). It had 1024 storage locations or memory addresses, called words. Each word is 40 bits long and may contain (1) a number word or (2) an instruction word (see Fig. 2).

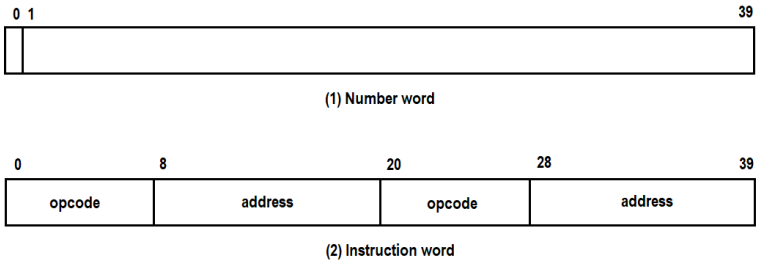

Fig. 2. Depiction of the two different types of words.

Instructions occupied two times 20 bits, where the first eight bits are opcode and the remaining twelve bits indicate the address of a register. Overall, the instruction set of the IAS machine contained 21 different instructions (Burks et al 1946, 42). A line of code of a program written for the machine then may look like this 00000001000111110100000001010001111101 01. The first eight bits (grey font) are opcode and correspond to the instruction "[c]lear accumulator and add number located at position x in the Selectron into it." (Load $M(x_i)$); the following twelve bits correspond to a memory address x; the next eight bits (grey font) are opcode and correspond to the instruction "[a]dd number located at position x in the Selectrons into the Accumulator" (Add $M(x_i)$). [20] It is sequences of bits like these, composed of the machines' specific instruction set that may comprise a program *P*.[21] As we will see, the reason why these details are relevant for the application of implementation-as is that they warrant multiple, distinct instances of interpretational exemplification.

Concerning the second main component, the CPU, the IAS machine has seven different registers (Accumulator, Arithmetic Register, Control Counter, Control Register, Function Table Register, Memory Address Register, and Selectron Register) of which only the Accumulator and the Arithmetic register are 'visible' to the programmer (both holding 40 bits).[22] These registers utilized about 1700 to 2300 commercially available miniature double triodes, [23] where most of them

[20] For a more elaborate and detailed example see for instance Priestley (2018).
[21] In the same vein, modern microprocessors are too compatible with specific ISAs (Instruction set architecture), like x86, where "[t]he ISA serves as the boundary between the software and hardware.", (Hennessy & Patterson 2012, 11).
[22] Today, the 'Control Counter' is known as Program Counter (12-bit width); the Control Register holds the instruction currently executing (20-bit width). The Function Table Register holds the current opcode and is 8 bits wide, whereas the Memory Address Register holds the current memory address and is 12 bits wide.
[23] The precise number of diodes used in the machine diverge among different authors. Whilst Estrin (1952) mentions 2300 triodes, Ware (1953) speaks of ca. 1700, and Bigelow (1980) mentions about 2000.

where of type 6J6 (other models used where 5670, 5687, and a few 6AL5 scattering diodes). Like modern garden variety CPUs, it executes instructions of programs, such as arithmetic (e.g., adding integers of above's example program *P*), I/O operations and logic controlling.

Lastly, the selected I/O components are an important element to consider. They afford the interface through which the users can interact and program the device. Without input mechanisms like punched cards, teletypewriters or keyboards, programmers and users had virtually no reliable means to load instructions or data into memory. In the same vein, the lack of an output medium (e.g., some kind of screen) would render the computational system a black box. It is these outputs however that ultimately need to be in tune with the denoted computational formalism/program *P*. At first, the engineers of the machine relied on perforated teletype tape which in late 1951 was replaced by IBM punched cards (Bigelow 1980, 306).

What turned the IAS-machine into a digital one is that it was operated under a digital encoding. This design choice both appealed to the intended logical nature of the machine ('being a yes-no system') and facilitated the use of existing electronic components (flip-flops), such that

> "[o]ur fundamental unit of memory is naturally adapted to the binary system since we don't attempt to measure gradations of charge at a particular point in the Selectron but are content to distinguish two states. The flip-flop again is truly a binary device.", (Brucks et al. 1946, 7).

In addition, the composition, or architecture, constituted by the three interconnected units M, CPU, and I/O enabled the IAS-machine to store instructions (and data) in memory. As such, the machine stands in contrast to early digital machines like ENIAC or analog devices like the MONIAC that had to be reprogrammed manually similar to plugboards or read instructions from external tape.

4.2. Implementation-as at work

Equipped with some basic understanding of the inner workings of the IAS-machine and how it was programmed, let me sketch how the most salient features of implementation-as come to fruition. As explained throughout the paper, the core notion of implementation-as is that properties of the designated computational vehicle are associated with the abstract computational states of a computational formalism $\{P_1, \ldots, P_m\}$ through a set of exemplified computational states $\{Z_{1, \ldots,} Z_n\}$. To implement a specific sequence of computation, subsequently the four steps of denotation, interpretational exemplification, encoding, and imputation need to apply to the putative computing system.

Here, we assume that the IAS-machine is our *X*, i.e., our vehicle of computation. As discussed in the previous section, our *X* is composed of many different components (e.g., cables, 6J6 triodes, …), forming three interconnected units (M, CPU, and I/O). As such, it can be considered a computing system under a series of fine-tuned interpretations *I* of some agent (typically the user of an epistemic community who share the same conventions regarding a device). Specifically, the IAS-machine then implements a computational formalism/program *P iff* the following four steps apply:

(1) First, the device *X* denotes *P*. In the case of the IAS-machine, a typical program *P* will look like a list of machine-code instructions each of 40-bit length as just introduced in the previous section. As such, *P* acts as the normative yardstick to evaluate executions between correct and

faulty ones (miscomputation). To eventually implement *P* correctly, different components of the IAS-machine need to relate to different sections of the code.

(2) Second, given our agreed upon interpretation *I,* we note that the IAS-machine exemplifies certain computational features $\{Z_{1, \ldots,} Z_n\}$. According to the general scheme outlined above, exemplification hinges on our interpretational capacities $I : X \rightarrow Z$. For instance, the previous discussion of the technicalities of the IAS-machine showed that the following components play different roles in exemplifying computational features: 5CP1A cathode ray tubes are employed for holding data and instructions in memory; the CPU (with its seven registers) relies on miniature triodes (mostly of type 6J6); the I/O used punched cards to program the machine in order code.

(3) Third, one needs to choose an encoding or labeling scheme. Since the IAS-machine was constructed as a binary digital computer, parting with the "longstanding tradition of building digital machines in the decimal system" (Brucks et al. 1946, 7), it operates as a binary digital computer processing both digital data and instructions in a binary format. Accordingly, we adopt a binary digital encoding as described in sect. § (3.3). Standardly, one then associates the absence (considering a certain threshold) of the flow of charge as '0' and the flow of charge as '1'.

(4) Finally, the just encoded computational states $\{P_1, \ldots, P_m\}$ are imputed to our 'targeted' program *P*. Since computer scientists, programmers and users usually opt for the correct implementation of computational artifacts, we ideally require that the entire set $\{P_1, \ldots, P_m\}$ is related to *P*.

To wrap up, the IAS-machine implements computations as the flow of charge. The straightforward and successful application of implementation-as to the IAS-machine strongly suggests that this novel agential theory of implementation may be equally well applied to other *bona fide* computers. Considerable technical differences withstanding, many modern computing machines still incorporate the basic architectural design choices of this influential device. I believe that it is sufficiently complex and bears enough similarities to the functioning of contemporary computers. While new technological advancements may induce ever more complexities, there is in principle no reason that would undermine the application of implementation-as to those cases.

5. Is Implementation-as a good theory of computation?

At last, let me briefly evaluate the in this article developed theory of agential implementation. The discussion proceeds along the lines of the desiderata of physical computation introduced in the introduction (Sect. 1). As I will show, implementation-as accommodates all the desiderata and should therefore be considered a viable theory of physical computation.

(1) Objectivity. Nowadays, philosophers of science commonly agree that there are considerable obstacles to cashing out theories of scientific representation in naturalistic terms. That is why most approaches are formulated as intentional conceptions (Frigg&Nguyen 2020a, 2020b). The DEKI account is a case in point, for all its salient features hinge on scientists' interpretational capacities. As discussed at length, implementation-as inherited many of the key features – and accordingly, it may be called an agential theory of implementation. Now, does relying on interpretational features undermine the objectivity of implementation-as?

The answer is nuanced. Reiss & Sprenger (2020) survey various conceptions of scientific objectivity – as stated by Fletcher (2018) and Duwell (2021), theories of physical computation based on agential notions of scientific representation may only undermine an overly rigid notion of

objectivity. Since implementation-as appeals to agents and their stipulations, it may be incompatible with what Duwell refers to as *strong objectivity* (i.e., an account of objectivity according to whether a system is representational/computational is completely mind-independent). However, relying on agential notions of scientific representation does not undermine *weak objectivity* (Duwell 2021, 19). Accordingly, scientists may reach intersubjective agreements if an object counts as a scientific model, which parts of the world it is presumed to represent, and so on. Once such intersubjective agreements are held fixed, practitioners may engage in scientific reasoning without their personal preferences or any substantial personal biases.

Implementation-as adheres to standards of objectivity in these latter, less rigid terms. Once the combined stipulative elements of denotation, interpretational-exemplification, encoding, and imputation are agreed upon and held fixed, computation under the regime of implementation-as is as objective as the scientific practice of modeling and free of personal arbitrary beliefs, desires, and intentions.

(2) Extensional Adequacy. A good theory of physical computation should properly systematize paradigmatic computing systems (laptops, calculators, smartphones) as computational; it should also judge instances of non-computing systems as non-computational. The examples of the MONIAC and the successful application to the IAS-machine show that implementation-as does not have trouble classifying paradigmatic examples of computing systems as computational. What works in the case of the IAS-machine, can then in principle be applied to other machines. In so far as the physical system exemplifies computational properties that are keyed-up/encoded and imputed to states of a computational formalism (which is denoted by the system), the system may implement the formalism as such and so.

However, frequently intentional conceptions of computation like implementation-as are rather deemed unsuccessful with respect to intuitively non-computational systems. So, for instance, without any restrictions on interpretational-exemplification, one might worry that every object could be turned into a computer by mere stipulation. Therefore, the claim that a system computes would be trivially true and hence uninformative. Traditionally, one option has been to simply bite the bullet and admit that, despite our intuitive understanding of paradigmatic computing systems, every system does indeed implement *some* computational functions (e.g., Chalmers 1996, Scheutz 1999).[24] While this leads to (at least) a form of limited pancomputationalism, one may still feel uneasy about such ubiquity of computation. Implementation-as comes with further restrictions though, limiting the extension of which objects may count as computational. Being informed by the literature on scientific representation, we draw from the corresponding parallel discussions of what may count as a scientific model. Two kinds of restrictions are chief; their combined interplay is depicted in Fig. 3. I already introduced them in section 3.2 on exemplification, where I described that both the choice of what may plausibly count as a model and analogously, as a computer, is constrained by two requirements – (i) exemplification under interpretation needs to be agreed upon and held fixed; and (ii) that the representation/computational vehicles need to offer counterfactual state transitions.

[24] This sort of pancomputationalism is limited, since, although it is postulated that every object computes, it does deny the much stronger thesis that these objects also implement every possible computation.

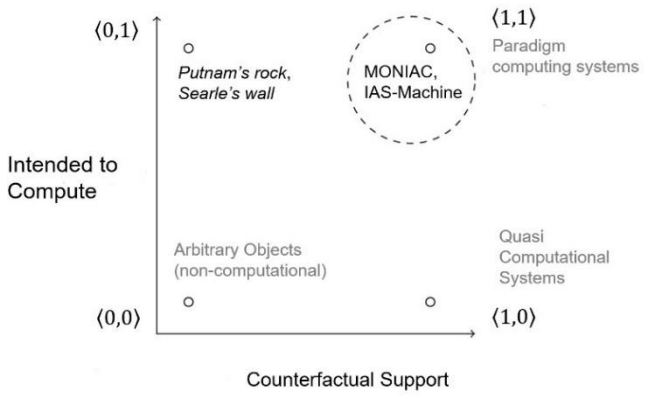

Fig. 3: The 'hybrid approach' (having to rely on both interpretation and the right degree of counterfactual support) ensures that implementation-as considers that the right things compute and the wrong things don't. Figure inspired by a similar graphic in Artiga (2023) in a different context (teleological functions).

(3) Explanation. According to the third desideratum, a good account of concrete computation should be able to explain (at least some of) a system's capacities computationally. There are different ways to understand this requirement. On the one hand, the computational properties of a system may be explained by what it implements. For instance, the IAS-machine implementing our exemplary program *P* explains why it adds integers the way it does, its efficiency, etc. Yet, on the other hand, under implementation-as material systems may only exemplify computational states if agents bestow them with the task to do so – without the agent's stipulations, the chosen vehicles are not computational. Does this mean that computational explanations then merely reduce to agents' desires to use something as a computer? No, because as I have argued implementation-as is a 'hybrid'-account – the agents also need to choose suitable physical states that may act as computational vehicles. That's why the current framework must additionally resort to the particular underlying scientific theories that describe the behavior of the chosen vehicles. As such, the explanations offered by implementation-as are no longer distinctively computational but may be physical, chemical, or biological (cf. Duwell 2021, 37). In the case of the MONIAC e.g., the flow of water is taken as a computational vehicle. To explain the behavior of the machine, we must consult hydrodynamics and the scientific theories describing the dynamics of the mechanical components.

(4) Miscomputation. One of the most basic advantages of interpretational accounts of computation is the straightforward explanation of judging the (in)correctness of a computational process. Unlike their naturalized counterparts that need to defend the vexing issue of natural teleology, interpretational accounts do not undermine their own metaphysical approach by maintaining that agents bestow the computing system with teleological functions to compute. Therefore, the philosophy of computer science accordingly borrowed some of the function ascription frameworks from the philosophy of technology (Turner 2018, Anderson 2019). So, as such an interpretational account of computation implementation-as can accommodate *different* notions of miscomputation. Let me briefly discuss these notions separately.

First, the programmers and users may disagree about which program is supposed to be implemented. Admittedly, this disagreement seems like a rather easily avoidable mistake. Nevertheless, denotation is still crucial for determining the (in)correct implementations of computations. For remember, figuring out the precise (teleological) function of a computing system is epistemically inaccessible, because it may not simply be read off. Prominent computer scientist Weizenbaum, for instance, brought up this inaccessibility in a thought experiment. If one day in the distant future a highly advanced society would find one of our present-day computers, they could never know with certainty to have gotten the alleged program *P* just right (Weizenbaum 1976, 132ff.). Albeit, a high degree of understanding might be achievable through observing its output patterns, black-box testing and attempts of reverse engineering, reclaiming absolute certainty of the computer's specification might be impossible.[25] Likewise, Dennett (1990) comes to a similar conclusion with a real-world example of a discovered 'computer' – the discovery of the *Antikythera mechanism*. When archaeologists lifted the ancient Greek hand-powered device from a shipwreck, the artifact's (teleological and mathematical) function was (at first) obscure and keeps scholars puzzled to the present day.

Secondly, miscomputation may be caused by faulty imputations. As argued above, faulty imputations may occur either through wear and tear or because of insufficient counterfactual support. Both conditions lead to a mismatch between the different execution traces of the denoted computational formalism M_C and the putative computing system.

(5) Taxonomy. Encoding a labeling scheme is crucial for determining for what kind of computations a system may be used for. I described the encodings corresponding to the arguably two most widespread instances of computing – digital and analog. Accordingly, the encodings of the interpreted computational vehicles enable us to discern two major kinds of computing systems and their different capacities.

Furthermore, implementation-as does not need to allude to the 'narrow' notion of program execution only. When judging various accounts of physical computation, Piccinini criticized some earlier approaches that would equate physical computation with program execution, because this may raise trouble for classifying systems that are said to compute by means other than running programs.[26] Implementation-as does not need to appeal to the notion of program execution in order to be applied successfully; nothing in its four salient features hinges on program execution. Rather, whether a system can be classified to compute by virtue of program execution depends on the denoted computational formalism (and arguably on one's definition of what a program is).

In sum, the results of this brief evaluation showed that implementation-as squares well with all the desiderata. It has everything what it needs to be called a good theory of implementation. What's more, the view has virtues that go beyond the five desiderata discussed above. Most notably, the

[25] Specifications are (formal) descriptions of computing systems. Specifications may be expressed in plain English (or any other natural language) or special (formal) specification languages like VDM or Z. In his conceptual analysis of specifications, Turner (2011) suggests that specifications are not merely descriptive, but also *prescriptive*. Typically, specifications come prior to the construction/programming of a device, subsequently determining constraints that a system must satisfy. In other words, specifications define which computational formalism is supposed to be implemented.

[26] For instance, he argues that some neural networks compute by means other than program execution (Piccinini2008). Another (potential) case in point is analog computers, where some scholars believe that they compute despite not executing a program.

account does not remain silent about the bridging problem of applied mathematics and is compatible with contemporary solutions of applied mathematics.

6. Conclusion

In this article, I presented a specific theory of agential implementation. This account extends on recent developments in the literature of physical computation, according to which one may appropriate the conceptual tools of scientific representation to computing. I corroborated this endeavor by in-depth details and providing a constructive perspective. Specifically, I relied on the insights of the DEKI account of material models and tweaked into the newly introduced conception called implementation-as. Since it is commonly accepted that scientific representation (and hence the DEKI account) is (partly) an intentional conception, implementation-as is too. Accordingly, the implementation of a computational function is dependent on the interplay of the stipulative capacities of users and the physical features of the material system. However, in lieu of advocating that concrete computation is the result of describing a physical system in terms of a computational model, the situation defended in this paper is reversed: Implementation comes about when material objects are utilized to 'model' abstract computational formalisms. Subsequently, agents may use a material computing system as a computing device if they engage in the combined activities of denotation, exemplification, encoding and imputation. While these four features need further specifications for each application, they encapsulate the commonalities of physical computation (two paradigmatic examples of an analog and a digital machine were discussed, viz., the MONIAC and the IAS machine).

As such, the upshot of the paper is twofold. First (and this was the main motivation for the paper), one may regard implementation-as as a vindication of the often-times shunned interpretational accounts of physical computation. Although interpretational accounts existed before, the here-promoted theory does not collapse into complete relativism/interpretational pancomputationalism. On the contrary, my analysis showed that implementation-as makes the grade with the standardly evoked desiderata for an adequate account of computation in physical systems. For these reasons, I conclude that implementation-as is a promising alternative to existing accounts of physical computation. Future research should investigate the framework's main tensions with research programs that demand a naturalized notion of computation, especially for the Computational Theory of Mind.

On a different note, the insights of the paper may serve as a gateway to start further research into the apparent conceptual similarities between modeling (especially with material systems) and computing.

References

(Anderson 2019) Anderson, Neal G. 2019. 'Information Processing Artifacts.' *Minds and Machines* 29 (2): 193–225.

(Artiga 2023) Antigua, Marc. 2023. 'A Dual-Aspect Theory of Artifact Function.' *Erkenntnis* 88 (4): 1533-1554.

(Aspray 1990) Aspray, William. 1990. *John von Neumann and the origins of modern computing. History of computing.* MIT Press.

(Barr 1988) Barr, Nicholas. 'The Phillips Machine' *LSE Quarterly*, 2(4): 305-337.

(Batterman 2010) Batterman, Robert W. 2010. 'On the explanatory role of mathematics in empirical science.' *The British Journal for the Philosophy of Science* 61 (1): 1–25.

(Bigelow 1980) Bigelow, Julian. 1980. 'Computer Development for the Institute of Advanced Studies', In *A History of Computing in the Twentieth Century: A Collection of Papers* (291–310) edited by Gian- Carlo Rota, N. Metropolis, J. Howlett. Academic Press, Inc.

(Bissel 2007) Bissell, C. 2007. 'Historical perspectives – The Moniac A Hydromechanical Analog Computer of the 1950s' *IEEE Control Systems Magazine*, 27(1): 59-64.

(Bokulich 2011) Bokulich, Alisa. 2011. 'How Scientific Models Can Explain'. *Synthese* 180 (1): 33–45.

(Bournez and Pouly 2021) Bournez, Olivier, and Amaury Pouly. 2021. 'A Survey on Analog Models of Computation', In *Handbook of Computability and Complexity in Analysis* (173–226), edited by Vasco Brattka and Peter Hertling. Cham: Springer International Publishing.

(Bueno and Colyvan 2011) Bueno, Otávio, and Mark Colyvan. 2011. 'An inferential conception of the application of mathematics'. *Noûs* 45 (2): 345–374.

(Burks 1980) Burks, Arthur W. 1980. 'From ENIAC to the Stored-Program Computer: Two Revolutions in Computers'. In *A History of Computing in the Twentieth Century: A Collection of Papers* (311–344), edited by Gian-Carlo Rota, N. Metropolis, J. Howlett. Academic Press, Inc.

(Burks et al. 1946) Burks, Arthur W., Herman H. Goldstine, and John von Neumann. 1946. *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument*.

(Callender and Cohen 2006) Callender, Craig, and Jonathan Cohen. 2006. 'There is no special problem about scientific representation. Theoria'. *Revista de teoría, historia y fundamentos de la ciencia* 21 (1): 67–85.

(Care 2010) Care, Charles. 2010. *Technology for modelling: electrical analogies, engineering practice, and the development of analogue computing*. Springer Science & Business Media.

(Chalmers 1996) Chalmers, David J. 1996. 'Does a rock implement every finite-state automaton?' *Synthese* 108 (3): 309–333.

(Copeland 1996) Copeland, B Jack. 1996. 'What is computation?' *Synthese* 108 (3): 335–359.

(Dennett 1990) Dennett, Daniel C. 1990. 'The interpretation of texts, people and other artifacts.' *Philosophy and phenomenological research* 50:177–194.

(Duwell 2021) Duwell, Armond. 2021. *Physics and Computation*. Cambridge University Press.

(Elgin 1983) Elgin, Catherine. 1983. *With Reference to Reference*. Hacket Publishing Company.

(Elgin 2010) Elgin, Catherine Z. 2010. in 'Telling Instances'. In *Beyond Mimesis and Convention: Representation in Art and Science* (1–17), edited by Mathew Hunter Roman Frigg. Springer.

(Elgin 2017) Elgin, Catherine Z. 2017. *True Enough*. Cambridge: MIT Press.

(Estrin 1952) Estrin, Gerald. 1952. A Description of the Electronic Computer at the Institute for Advanced Studies. *Proceedings of the 1952 ACM National Meeting (Toronto)*, ACM '52. New York, NY, USA: Association for Computing Machinery, 95–109.

(Fodor 1981) Fodor, Jerry. 1981. 'The mind body problem' *Scientific American* 244 (1): 114–125.

(Fletcher 2018) Fletcher, Samuel C. 2018. 'Computers in Abstraction/ Representation Theory.' Minds and Machines 28 (3): 445–463.

(Fresco2014) Fresco, Nir. 2014. *Physical computation and cognitive science*. Springer.

(Fresco and Primiero2013) Fresco, Nir, and Giuseppe Primiero. 2013. 'Miscomputation'. *Philosophy & Technology* 26 (3): 253–272.

(Frigg and Nguyen 2017) Frigg, Roman, and James Nguyen. 2017. 'Scientific Representation Is Representation-As'. In *Philosophy of Science in Practice: Nancy Cartwright and the Nature of Scientific Reasoning* (149–179), edited by Hsiang-Ke Chao and Julian Reiss. Cham: Springer International Publishing.

(Frigg and Nguyen 2018) Frigg, Roman, and James Nguyen. 2018. 'The turn of the valve: representing with material models.' *European Journal for Philosophy of Science* 8 (2): 205–224.

(Frigg and Nguyen 2020a) Frigg, Roman, and James Nguyen. 2020a. *Modelling nature: An opinionated introduction to scientific representation*. Springer.

(Frigg and Nguyen 2020b) Frigg, Roman, and James Nguyen. 2020b. 'Scientific Representation'. In *The Stanford Encyclopedia of Philosophy*, edited by Edward N. Zalta, Winter 2016. Metaphysics Research Lab, Stanford University.

(Giere1999) Giere, Ronald N. 1999. 'Using models to represent reality.' In *Model-based reasoning in scientific discovery* (41–57), edited by L. Magnani, N. Nersessian, and P. Thagard. Springer.

(Godfrey-Smith2009) Godfrey-Smith, Peter. 2009. 'Triviality arguments against functionalism'. *Philosophical Studies* 145 (2): 273–295 (Aug).

(Goodman1976) Goodman, Nelson. 1976. *Languages of Art: An Approach to a Theory of Symbols*. Hackett Publishing Company, Inc.

(Hennessy and Patterson2012) Hennessy, John L, and David A Patterson. 2012. *Computer architecture: a quantitative approach*. 5. Elsevier.

(Horsman et al. 2014) Horsman, Clare, Susan Stepney, Rob C Wagner, and Viv Kendon. 2014. 'When does a physical system compute?' *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 470 (2169): 20140182.

(Horsman 2017) Horsman, Dominic. 2017. 'The representation of computation in physical systems.' In *EPSA15 selected papers*, 191–204. Springer.

(Horsman, Kendon, and Stepney 2018) Horsman, Dominic, Viv Kendon, and Susan Stepney. 2018. 'Abstraction/ Representation Theory and the Natural Science of Computation'. In *Physical Perspectives on Computation, Computational Perspectives on Physics* (127–150), edited by Michael E. Cuffaro and Samuel C. Editors Fletcher. Cambridge University Press.

(Horsman et al. 2017) Horsman, Dominic, Viv Kendon, Susan Stepney, and J Peter W Young. 2017. 'Abstraction and representation in living organisms: when does a biological system compute?' In *Representation and reality in humans, other living organisms and intelligent machines*, (91–116), edited by Gordana Dodig-Crnkovic and Raffaela Giovagnoli. Springer.

(Horsman, Kendon, and Stepney 2017) Horsman, Dominic, Vivien Kendon, and Susan Stepney. 2017. 'The Natural Science of Computing'. *Commun. ACM* 60 (8): 31â34 (July).

(Horsman 2015) Horsman, Dominic C. 2015. Abstraction/representation theory for heterotic physical computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 373 (2046): 20140224.

(Hughes 1997) Hughes, Richard. 1997. 'Models and Representation'. *Philosophy of Science* 64 (4): 336.

(Klein 2008) Klein, Colin. 2008. 'Dispositional implementation solves the superfluous structure problem'. *Synthese* 165 (1): 141–153.

(Kripke 1982) Kripke, Saul A. 1982. *Wittgenstein on rules and private language: An elementary exposition.* Harvard University Press.

(Ladymen 2009) Ladymen, James. 'What does it mean to say that a physical system implements a computation?' *Theoretical Computer Science* 410, 376–383.

(Fortune 1952) Unknown author. 'The Moniac'. *Fortune* March 1952, 101.

(Lewis 1971) Lewis, David K. 1971. 'Analog and Digital'. *Noûs* 5 (3): 321–327.

(Maley 2011) Maley, Corey J. 2011. Analog and Digital, Continuous and Discrete. Philosophical Studies 155 (1): 117–131.

(Marr 2010) Marr, David. 2010. *Vision: A computational investigation into the human representation and processing of visual information*. MIT Press.

(Miłkowski2013) Miłkowski, Marcin. 2013. *Explaining the computational Mind*. MIT Press.

(Morgan 2012) Morgan, Mary S. 2012 *The World in the Model*. Cambridge University Press.

(Morgan and Boumans 2004) Morgan, Mary S. and Marcel Boumans. 2004. 'Secrets Hidden by Two-Dimensionality: The Economy as a Hydraulic System'. In *Models: The Third Dimension of Science* (369– 401). Stanford University Press.

(Newlyn 1950) Newlyn, Walter T. 1950. 'The Phillips/Newlyn Hydraulic Model' *Yorkshire Bulletin of Economics* 17: 282–305.

(Nguyen and Frigg 2017) Nguyen, James, and Roman Frigg. 2017. 'Mathematics is not the only language in the book of nature'. *Synthese*, pp. 1–22.

(Phillips 1950) Phillips, Alban W. H. 1950. 'Mechanical Models in Economy Dynamics' *Economia* 18 (67): 283–305.

(Papayannopoulos 2020) Papayannopoulos, Philippos. 2020. 'Computing and modelling: Analog vs. Analogue'. *Studies in History and Philosophy of Science Part A* 83:103–120.

(Piccinini 2007) Piccinini, Gualtiero. 2007. 'Computing Mechanisms'. *Philosophy of Science* 74 (4): 501–526.

(Piccinini 2008) Piccinini, Gualtiero. 2008. 'Some neural networks compute, others don't' *Neural Networks* 21 (2008): 311–321.

(Piccinini 2015) Piccinini, Gualtiero. 2015. *Physical computation: A mechanistic account*. Oxford University Press.

(Pincock 2004) Pincock, Christopher. 2004. 'A new perspective on the problem of applying mathematics'. *Philosophia Mathematica* 12 (2): 135–161.

(Pincock 2022) Pincock, Christopher. 2022. 'Concrete Scale Models, Essential Idealization, and Causal Explanation'. *British Journal for the Philosophy of Science* 73 (2): 299–323.

(Pour-El 1974) Pour-El, Marian Boykan. 1974. 'Abstract Computability and Its Relation to the General Purpose Analog Computer (Some Connections Between Logic, Differential Equations and Analog Computers)'. *Transactions of the American Mathematical Society* 199:1–28.

(Priestley 2018) Priestley, M. 2018. *Routines of Substitution: John von Neumann's Work on Software Development, 1945–1948*. Springer-Briefs in History of Computing. Springer International Publishing.

(Psillos 2006) Psillos, Stathis. 2006. 'The Structure, the Whole Structure, and Nothing but the Structure?' *Philosophy of Science* 73 (5): 560–570.

(Putnam 1988) Putnam, Hilary. 1988. *Representation and Reality*. MIT Press.

(Reiss and Sprenger2020) Reiss, Julian, and Jan Sprenger. 2020. 'Scientific Objectivity'. In *The Stanford Encyclopedia of Philosophy*, edited by Edward N. Zalta, Winter 2020. Metaphysics Research Lab, Stanford University.

(Ritchie and Piccinini 2018) Ritchie, J. Brendan, and Gualtiero Piccinini. 2018. 'Computational implementation'. In *The Routledge Handbook of the Computational Mind.* (192–204), edited by Mark Sprevak and Matteo Colombo. Routledge.

(Scheutz 1999) Scheutz, Matthias. 1999. 'When physical systems realize functions...'. *Minds and Machines* 9 (2): 161–196.

(Schweizer 2019) Schweizer, Paul. 2019. 'Computation in physical systems: A normative mapping account'. In *On the cognitive, ethical, and scientific dimensions of artificial intelligence*, 27–47. Springer.

(Shagrir 2020) Shagrir, Oron. 2020. 'In Defense of the Semantic account' *Synthese* 197 (9): 4083–4108.

(Sprevak 2018) Sprevak, Mark. 2018. 'Triviality arguments about computational implementation'. In *The Routledge Handbook of the Computational Mind* (175–191), edited by Mark Sprevak and Matteo Colombo Routledge.

(Suárez 2003) Suárez, Mauricio. 2003. 'Scientific Representation: Against Similarity and Isomorphism'. *International Studies in the Philosophy of Science* 17 (3): 225–244.

(Suppes 2002) Suppes, Patrick. 2002. *Representation and Invariance of Scientific Structures*. CSLI Publications.

(Swoyer 1991) Swoyer, Chris. 1991. 'Structural Representation and Surrogative Reasoning'. *Synthese* 87 (3): 449–508.

(Szangolies 2020) Szangolies, Jochen. 2020. 'The Abstraction/ Representation Account of Computation and Subjective Experience.' *Minds and Machines* 30 (2): 259–299.

(Turner 2011) Turner, Raymond. 2011. 'Specification'. *Minds and Machines* 21 (2): 135–152 (May).

(Turner 2018) Turner, Raymond. 2018. *Computational Artifacts*. Springer.

(Ulmann 2013) Ulmann, Bernd. 2013. *Analog computing*. Oldenburg Verlag.

(Ulmann 2020) Ulmann, Bernd. 2020. *Analog and hybrid computer programming*. Walter de Gruyter GmbH & Co KG.

(van Fraassen 2008) van Fraassen, Bas C. 2008. *Scientific Representation: Paradoxes of Perspective*. Oxford University Press.

(von Neumann 1945) von Neumann, John. 1945. *First Draft of a Report on the EDVAC*

(Ware 1953) Ware, Willis H. 1953. 'The History and Development of the Electronic Computer Project at the Institute for Advanced Study'. Santa Monica, CA: RAND Corporation.

(Weisberg2013) Weisberg, Michael. 2013. *Simulation and similarity: Using models to understand the world*. Oxford University Press.

(Weizenbaum 1976) Weizenbaum, Joseph. 1976. *Computer Power and Human Reason: From Judgment to Calculation*. USA: W. H. Freeman Co.

(Wiggershaus 2023) Wiggershaus, Nick. 2023. *Towards a Unified Theory of Implementation.* Preprint. (http://philsci-archive.pitt.edu/id/eprint/22100)