Review

Not peer-reviewed version

# On Implementing Technomorph Biology for Inefficient Computing

Végh János *

Posted Date: 28 February 2025

doi: 10.20944/preprints202502.2238.v1

Keywords: biological computing; technical computing; biomorph architectures; computing efficiency; technical performance; cost function of computing

*Article*

# On Implementing Technomorph Biology for Inefficient Computing

**János Végh**

Kalimános Bt; vegh.Janos@gmail.com

**Abstract:** It is commonly accepted that 'The brain computes' and that it served as a model for establishing principles of technical (first of all, electronic) computing. Even today, some biological implementation details inspire the implementation of more performant electronic implementations. However, grasping out details without context often leads to decreasing operating efficiency. In the cases of those major implementations, the notion of 'computing' has an entirely different meaning. We provide the notion of generalized computing from which we derive technical and biological computing, and by showing how the functionalities are implemented, we also highlight what performance losses lead the solution. Both implementations have been developed using a success-failure method, keeping the successful part-solutions (and building on top of them) and replacing a less successful one with another. Both developments proceed from a local minimum of their goal functions to another, but some principles differ fundamentally. Moreover, they apply entirely different principles, and the part-solutions must cooperate with others, so grasping some biological solution without understanding its context and implementing it in the technical solution usually leads to a loss of efficiency. Today, technical systems' absolute performance seems to be saturated while their computing and energetic inefficiency are growing.

**Keywords:** biological computing; technical computing; biomorph architectures; computing efficiency; technical performance; cost function of computing

---

## 1. Introduction

Concerning computing performance, efficiency, and power, we have a high standard, a desired goal, and an existing prototype: "The human brain is . . . the most powerful and energy-efficient computer known to humankind." [1] However, when attempting to imitate its operations, the fundamental difficulty is that "engineers want to *design systems*; neuroscientists want to *analyze an existing one* they little understand" [2]. Despite the impressive results of grandiose projects [1,3], *"we so far lack principles to understand rigorously how computation is done* in living, or active, matter" [4]. Even within biology, the too-close parallels with the electric operation are at least questionalized [5] (given that the model is called "realistic"): "Is realistic neuronal modeling realistic?" Engineers grasped a few aspects of the active matter's operation and implemented those aspects without its context within the frames of their possibilities. Implementing "lacking the principles of understanding how computation is done in living matter" in the different contexts of technical computing is dangerous. It leads to solutions that are neither biological nor technical.

A typical example of misunderstanding the role of the most evident phenomenon in neural systems is spiking. Many publications use the idea of "spiking neural networks". As we discuss in Section 4, spiking is just a technical tool nature uses to transmit information. Although it is well known from communication theory [6] that a single spike cannot deliver information, the so-called neural information theory extensively uses the fallacy [7–9] that firing rate delivers information. Answering the fundamental question is abandoned: how the information is represented in neurons, in their networks, and the transmission signal called spike. Despite that, the relevance of the "coding metaphor" is at least questionable for biology [10]; different coding systems are created based on hundreds of communication models [7]; not related to Shannon's model, but referring to its theory. [11] "What makes the brain a remarkable information processing organ is not the complexity of its neurons but the fact that it has many elements interconnected in a variety of complex ways." [12], page 37.

Similarly, definitions such as "Multiple neuromorphic systems use spiking neural networks (SNNs) to perform computation in a way that is inspired by concepts learned about the human brain" [13] and "SNNs are artificial networks made up of neurons that fire a pulse, or spike, once the accumulated value of the inputs to the neuron exceeds a threshold" [13] are misleading. That review "looks at how each of these systems solved the challenges of forming packets with spiking information and how these packets are routed within the system". In other words, the technically needed "servo" mechanism, a well-observable symptom of neural operation, inspired the "idea of spiking networks". The idea lacks knowing neurons' information handling, replaces the (in space and time) distributed information content by well-located timestamps in packets, the continuous analog signal transfer by discrete packet delivering, the dedicated wiring of neural systems by bus-centered transfer with arbitration and packet routing, the continuous biological operating time with a mixture of digital quanta of computer processing plus transferring time. Even the order of timestamps conveyed in spikes (not to mention the information they deliver) is not resemblant to biology; furthermore, due to the vast need for communication, the overwhelming majority of the generated events must be killed [14] without delivering and processing them, to provide a reasonable operating time and to avoid the collapse [15] of the communication system. In the rest of the aspects, the system may provide an excellent imitation of the brain.

We review how the computing demands govern the computing industry toward implementing higher computing capacity using different theoretical principles and technological solutions for different computing goals, while the cost functions such as energy consumption, computing speed, fabrication cost, scalability, reliability, etc., change highly. We summarize the generalized principles of computing in Section 2 and derive how those principles can be implemented by technical and biological computing. We also show that any computing process inherently includes inefficiency and derive the terms for describing it quantitatively. For technical computing, we only mention how those generic events are implemented and how the features of biological computing shall be interpreted in terms of technical computing. For biological computing, in Section 4, we derive a conceptual model in technical terms (without mentioning the biological details), emphasizing the consequences of how those events are generated. We emphasze the vital subject of data transmission (communication) between elementary computing units in chained operations. In light of the fundamental discrepancies, we discuss why the popular notions of learning and intelligence have only their name in common. Furthermore, we demonstrate how the wrong understanding of biological principles degrades the computing efficiency of technical systems.

## 2. Notions of Computation

"Devising a machine that can process information faster than humans has been a driving force in computing for decades" [11] since "computation is a useful concept far beyond the disciplinary boundaries of computer science." [4]. We aim to broaden this concept by applying it to technical and biological systems.

"Research shows that generating new knowledge is accomplished via natural human means . . . this knowledge is acquired via scientific research" [16], including various forms of cooperation of those two kinds of computing. Their notions have been attempted to map to each other. The human-constructed computing introduced notions such as bit (including its information content), structural and functional units for computing, elementary computing units implementing mathematical functions, and storing and transmitting information in special devices; those terms were not known in the nature-constructed biological computing [17]. It is *questionable whether nature must accommodate its computing-related notions to human-made computing*.

### *2.1. the Fundamental Task*

"Computation is a useful concept far beyond the disciplinary boundaries of computer science." [4] When wanting to perform a computation, one needs to assign operations to the input data [18] in various forms, from verbal instructions to wiring different electronic computing units by cables to setting a program pointer to address the first instruction of a stored program. Usually, the operations must be chained. Practical computations comprise many elementary operations, supplying input data for each other. Their signal representation and path, sequencing, and timing shall be provided to chain them appropriately. Data transfer and processing must not be separated; they must be discussed together. For geared wheels, biological, neuromorphic, quantum, digital, analog, neuronal, and accelerated processing, the operation can only begin after its required input operands are transferred to the place of operation, and the output operand can only be delivered once the computing process terminates. Our paper applies this concept to technical and biological systems, highlighting its universal significance. The model of computing did not change during the times, for centuries:

- input operand(s) need to be delivered to the processing element
- processing must be wholly performed
- output operand(s) must be delivered to their destination

Correspondingly, the effective computing time comprises the sum of the transfer time(s) and the processing time, even if, in the actual case, one can be neglected apart from the other. (Interestingly, biology uses point-like neurons and omits processing time aside from axonal delivery time, while technical computing omits transfer time aside from processing time, despite von Neumann's warning.) *Phases of computing logically depend on each other and must be appropriately aligned*. In their implementations, they need proper synchronization. In this way, *the data transfer and processing phases block each other*. That is, *any technological implementation converts the logical dependence of their operations to temporally adjacent phases* (used to call it von Neumann-style computing [19]): *to perform a chained operation, one's output must reach the other's input*. This definition enables multiple elementary operations to be performed in the second step, provided that they fulfill the other two requirements individually. This aspect deserves special attention when designing and evaluating the correct operation of computing accelerators, feedbacks, and recurrent circuits, as well as when designing and parallelizing algorithms: the computation considers the corresponding logical dependence through their timing relations [20].

The principle speaks only about sequential chained operations. However, it enables multiple operations to be performed simultaneously, provided that the synchronization points between the steps are kept (meaning that the next operation can only start when the slowest operation is finished). Looping elementary operations is only allowed when some independent method guarantees that the result of an operation performed later time will be replaced by an appropriately good value for the actual operation. This latter need may remain hidden for cyclic operations but might come to light by causing drastical effects on the computation (say making the otherwise correct initial or previously calculated values wrong), for example, when beginning a computation or changing one operation type to another.

### *2.2. Modeling Computation*

To perform a computation, one needs to assign operations to the input data, whether the computation is manual or electronic [21]. Practical computations comprise many elementary operations, supplying input and output data for each other: the operations must be chained. Either the instruction looks up its associated operand (instruction-driven control), or the data looks for its associated instruction (data-driven control). Data transfer and processing must not be separated; they must be discussed together. Anyhow, their way of correspondence, their sequencing and signal representation, path, and timing shall be provided to chain the operations appropriately. *Events* must signal the computation's milestones (such as its beginning and the result's availability). Von Neumann's famous *omission of the transfer time aside from the processing time in chained computing operations* was valid only for [the timing relations of] vacuum tubes. He clarified that using "too fast" vacuum tubes (or other elementary processing switches) *vitiates his simplified paradigm that forms the basis of today's computing science*.

The first operating unit of the chain is given the input data that operates the computing chain, and the last unit provides the result. A large number of elementary computing objects shall be provided for more sophisticated and lengthy computations, or some units can be reused after making an appropriate organization. The first option means wasting computing components, enabling a considerable parallelization for short instruction sequences, but its operation needs nearly no organization. The second option requires a higher-level organizer unit, enables minimum parallelization (presumes that the instructions use some computing units exclusively), and allows for an unlimited operating time and computing operations. However, either operating mode needs a *timing alignment* for the correct operation.

## 2.3. Time Windows

The times of the starting and terminating *event*s mark a *time window* in which the data processing happens; the window width is the time needed to perform the computing operation. Biology and technology mark the time windows differently. The position and width of the time window are key to the operation of the computations. In technical computations, the clock signal defines the position, while in biological computations, the time of the first input.

Technology commonly imitates those events and periods by using a *fixed-time central synchrony* signal. One edge of the signal defines the beginning, and another one the termination of the operation, see Figure 1 (the figure shows a symbolic notation for aligning events instead of clock edges). At the beginning, all input data must already be available in the input section of the unit, and the result must appear at its output before the time window terminates [22]. The designer is responsible for delaying the 'Begin Computing' signal at the start of the computation (*leading to performance loss*) for the clock signal because both signals have finite propagation speeds. In other words, *all timings and lengths of operations must be known at design time*, and they cannot be fine-tuned at run time. All operations have the same temporal length (allowing for pipelining). If the operation completes before the time window terminates, the remaining time gets lost, leading to *performance loss*. If the operation is incomplete, the chained unit may work with garbage. The elementary operations implement a mathematical function, and the elementary computing units are self-contained. We can define *the total length* of the operation as the sum of the computing and transferring windows' lengths.

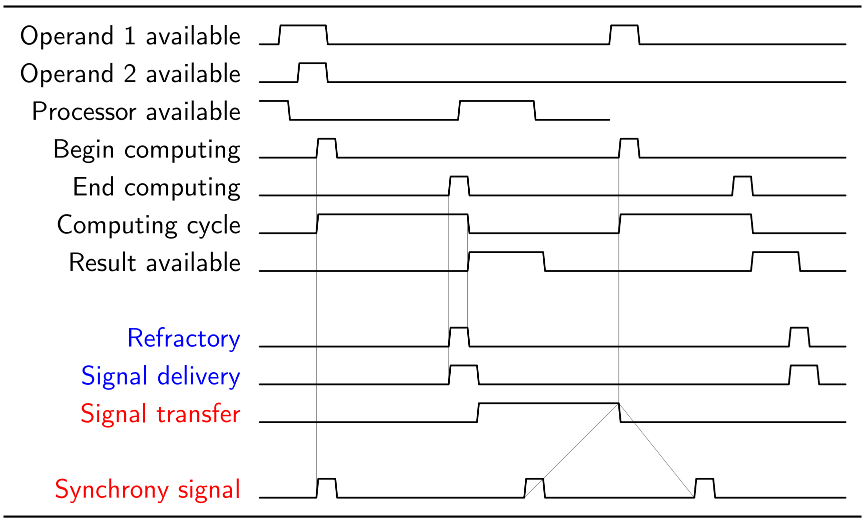

**Figure 1.** Timing relations of von Neumann's model for chained computation

The transfer time window starts when a unit produces its result and terminates at the time when the transferred data arrives at the next computing element. Instead of the previous calculation, the clock signal starts the unit, leading to further *performance loss*. Technology has various implementations, from direct galvanic connection to arbitrated sequential transfer over the bus to using special network transfer devices. *Time windows with very different widths belong to those modes* (ranging from psecs to msecs). No separate transfer time window is provided, so the transferred signal must adapt to the central clock signal. *As predicted by von Neumann, under the varying timing relations of modern technology, theory cannot describe the computing operations precisely*. The computing model itself, however, is correct, and the general computing paradigm [21] is appropriate for describing biological and technical computing, provided that the finite interaction speed is used instead of the classical instant interaction.

*These time windows must not overlap: it would lead to causality issues*. Data arriving from a more considerable distance or needing extended processing causes delays, blocks computing operations, and decreases computing efficiency. In the case of simple chained operations, this condition can be controlled, but ad-hoc (mainly performed in multiple steps) parallelizations and loops (feedback, back-propagation, accelerators) need great care. They can lead to errors [21]. In those latter cases, *the starting and the terminating data processing times are unknown at design time; however, the operation of the system's components must consider central synchronization (i.e., an external constraint), and it must use fixed processing times*.

Figure 1 shows the timing relations of the complete (time aware) model (it is a logical summary instead of a precise timing diagram). Notice that in chained computing operations using central synchrony signals, *the total computing times must be as uniform as possible instead of the processing times*. If the operation's *processing times* (complexity) or *transfer times* (connection technology, including different package handling, or signal's physical propagation speed or distance the signal has to pass) differ, the *total computing time* changes. It is worth noticing that the different timing contributions and their effect sensitively affect the system's resulting performance.

Using time windows seem to be a general biological mechanism. *The window's width depends on the task's complexity: more involved neuron levels (whether natural or artificial) need more time to deliver the result. The individual neurons transfer data in a massively parallel way, but the levels must wait until the result of the previous level arrives.* The individual cells operate in the msec range; they cooperate in several msecs [23,24], implement Hebbian learning in tens of milliseconds [25], perform cognitive functions in several hundred msecs [26] and student learn in time windows of dozens of seconds [27].

## *2.4. Three-Stage Computing*

An often forgotten requirement is that, theoretically, a three-state system is needed to define the direction of time [28]. In electronic computing, we can introduce this as an "up" edge and "down" edge, with a "hold" stage between. A charge-up process must always happen before discharging. As discussed below, biological computing has a computing stage, followed by "up" and "down" edges; always in this order. This requirement is a fundamental issue for quantum-based computing, where the processes are reversible: we must define the direction of time; otherwise, only random numbers can be generated.

## *2.5. Issues in Formulating Efficiency*

The false parallels with electric circuits (i.e., neglecting the fundamental differences between the *digital* and *neuro-logical* operating modes), moreover, the preconception that nature-made biological computing must follow notions and conceptions of manufactured computing systems (see, for example, the electrotonic models [29], where one attempts to describe the properties of neurons assuming that they are electronically manufactured components) hinders understanding genuine biological computing. Despite those difficulties, brave numerical comparisons exist, even with the so-called "Organoid Intelligence" [30].

The "Brain initiative", after ten years, had to admit [1] that "yet, for the most part, we still do not understand the brain's underlying computational logic." [1] The Human Brain Project believes that it is sufficient to build a computing system with resources [31] sufficient for simulating 1 billion neurons, and does not want to admit that it can be used to simulate only by orders of magnitude less [14], below 100 thousand. The presently available serial systems do not enable the performance to be reached [32]. The project is declared successful even though the targeted “studies on processes like plasticity, learning, and development exhibited over hours and days of biological time are outside our reach” [14]. Despite the lack of knowledge, the half-understood principles of neuromorphic computing [33] are extensively used [33,34], although it is seen that “brain-inspired computing needs a master plan” [35]. However, the hype is enormous [33].

It isn’t easy to compare technical and biological computing systems, either their computing ability or energetic efficiency. In their infancy, researchers saw many similarities between "the computer and the brain" [36]. We experience that "The brain computes!" [37], and the idea about their similarity was periodically warmed up and revisited [38,39]. Also, some resemblance is periodically claimed between natural and artificial intelligence. When one attempts to compare computing by the brain to computer computing, the first principal problem is assessing the brain’s computing power. One can "define an elementary operation of the brain as a single synaptic event" [40], and its operands and result are *analog timing values*. In contrast, "in a digital computer, an elementary operation constitutes . . . a few instructions such as load, add, and multiply" [40] using *digital signals*. It is at least problematic to compare the elementary operations by computing power or speed.

The second is how to map these operations to each other given that thousands of machine instructions are needed to emulate an analog integration (diffusion) and thousands of neurons are needed to perform simple digital addition, not to mention the thousands of machine instructions required to connect to the external world [41–43]; so the question how to map these operations to each other remains open. In reality, the biological data transmission time is up to two orders of magnitude longer than the computing time. In technical computing, the transmission through a bus might result in similar ratios. Furthermore, the operations can be at least partly parallelized. “The naïve conversion between OPS and SOPS as follows: 1 SOPS $\approx$ 2 OPS” [44] has limited validity. It can differ from the real one by two to four orders of magnitude in both directions, depending on the context, not to mention cross-comparison between biological and technical implementations. Any conversion reflects the misunderstanding that the total operating time comprises only the computing time, and the data transmission time can be neglected.

The third problem we face is the "non-deterministic", non-perfect, analog, asynchronous way of operation of the brain versus the deterministic, perfect, digital, synchronous operation of the computer. Furthermore, the arrival time of the received postsynaptic spikes [45] almost entirely determine the input arguments. We also know that the brain has no central clock, instruction pointer, or memory unit; furthermore, it uses mixed data-flow and event control and learns excellently.

The fourth one is that a neuron is "a multiaccess, partially degraded broadcast channel that performs computations on data received at thousands of input terminals and transmits information to thousands of output terminals by means of a time-continuous version of pulse position" [46] and works with information distributed in space and time. At the same time, technical communication applies Shannon’s point-to-point communication [6] model and works with discrete digital signals.

Fifth, the brain works with slow and heavy ions; furthermore, it is governed by ’non-ordinary’ laws [47] describing mixed slow thermodynamical and fast electrical interactions [48]. In contrast, technical computing is governed by the well-known laws of electricity, and non-sequential technical signal processing times contribute to its ’theoretical’ processing times.

As [35] classified, “the term neuromorphic encompasses at least three broad communities of researchers, distinguished by whether their aim is to emulate neural function (reverse-engineer the brain), simulate neural networks (develop new computational approaches), or engineer new classes of electronic device.” However, all these communities inherited the wrong ideas above.

Their operating principle, logical structure, architecture, signal representation and propagating speed, synchronicity and communication methods, and many other biological and technical computing aspects are fundamentally different. We have classical digital computers with (in the above sense, limited validity) comparable capacity and the brain itself (used many times as etalon for comparisons),

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 February 2025 doi:10.20944/preprints202502.2238.v1

plus an uncountable variety of so-called “neuromorphic” computing devices (including partly biological ones) providing a smooth transition between, furthermore fully biological “organoid” systems (such as [30,49]. We attempt to compare their elementary operations (serving as a base for their highest level of operation, “intelligence”,) in a strictly technical sense at the level of elementary operations without touching philosophical, mathematical, and ethical questions.

As we discuss, those implementations have inherent, deeply rooted technology solutions. Of course, technological enhancements are needed and valuable, but they only enhance improper technological solutions without understanding the computing theory. Moreover, computing lacks the appropriate theory. What is commonly considered a theoretical base was validated by von Neumann[50] to technical conditions of vacuum tubes. As he predicted, the technical development invalidated his omissions introduced to simplify the initial designs.

Figure 2 shows how the loss due to omitting the transfer time (and due to that, omitting the need of agreeing temporal length of different logical paths) leads to hidden performance loss. The gates provide outputs for each other, so to start some activity, they must wait until all needed signals, including the result produced by the fellow gates, reach their input section. In the figure, the vertical axis shows the times when the needed logical activities happen at the gates. The locations of input signals and operations are shown on different axes for visibility. The red vectors show where the adder circuit is waiting (a non-payload activity).

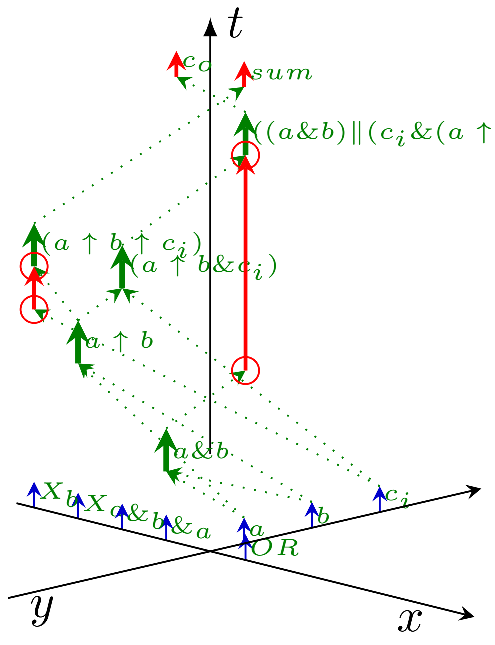

**Figure 2.** Idle times in a simple 1-bit adder.

It cannot successfully grasp and implement an idea from biology without understanding its context, underlying solutions, and requirements. Without understanding the inspiring object’s operation, implementing it using already developed, available materials, technology, components, and principles can show initial successes (mainly in “toy” systems). However, it can enable the development of neither even tiny functionally equivalent parts of the brain nor biomorphic general-purpose computers, not to mention their computational efficiency. This is why judges of the Gordon Bell Prize noticed that "surprisingly, [among the winners of the supercomputer competition] there have been no brain-inspired massively parallel specialized computers" [51]. Computing principles (including its fundamental theory) need rebooting [52,53] (a leap-like development) instead of incremental enhancement of some of its components and manufacturing technology solutions.

We can only admire von Neumann’s genial prediction that "the language of the brain, not the language of mathematics" [36], given that most of the cited neuroscience experimental evidence was unavailable at his age. We also agree with von Neumann and Sejnowski that “whatever the system [of the brain] is, it cannot fail to differ from what we consciously and explicitly consider mathematics." [40] However, the appropriate mathematical methods may not yet be invented.

## 2.6. Payload Vs. Theoretical Efficiency

We define *computing efficiency*, based on the execution times, as the ratio of the ’net computing’ time to the ’total operating’ time from utilizing the computing machinery. The ’net’ computing time is the period our ’black box’ elementary computing unit needs after we make its needed arguments available through its input unit and generate the event ’Begin computing’ until it produces its result in its output section by generating the event ’End computing’. Since the data must be transferred from one unit’s output section to another’s input section, even if physically the same computing unit is used, chained computing inherently loses efficiency, given that transferring data has a finite (although large) speed. An operating time shorter than the clock period, a transfer time longer than the clock period, a delay of clock edges, or any other timing issue, decreases *payload efficiency*. In contrast, the theoretical(ly possible) efficiency and raw computing power, remain the same. When the operations follow each other without a break (i.e., the data transmission time can be omitted aside from the computing time, and all computing units receive in time the arguments they need to compute with), we have a 100% (or theoretical) efficiency. The payload efficiency depends on the system’s operating mode, construction, and workload [54].

Another way of defining efficiency is by determining the percentage of the available computing facilities used in the computation in the long run. Another way is to determine the percentage of power used directly for computation versus indirect computing (organizing the work or making non-payload computation) or the total power or the power needed for communication. Using a precise definition is important. When multiple-issue computing units are present but only part of them are used (say by out-of-order execution), or their result becomes obsolete (say by branch prediction), computing throughput (and power consumption) increases, but utilization of the available hardware decreases. Similarly, as we discuss in connection with deep learning (or other nonsense feedback), making obsolete and nonsense operations reduces computing efficiency.

For the commonly used instruction-driven sequential systems, the computing time equals the total time only if the transfer time can be omitted and all computing units can be fully utilized. That was the case for the vacuum tube implementation and a single computing unit: the transfer time was in the *μsec* region, while the operation time and memory access time were in the *msec* region [50]. (Notice that when mentioning the so-called ’von Neumann bottleneck,’ the stored program concept, one must oppose the time of manually wiring the computing units to load the program in advance and have memory access per instruction).

## 2.7. Instruction- and Data-Driven Modes

In a computation, data, instructions, and their way of correspondence must be provided. Either the instruction finds its associated operand (instruction-driven control, usually implemented by fetching stored instruction given by their address), or the data finds its associated instruction (data-driven control, usually by wiring). In the case of data-driven operation, all data have their exclusively used pre-wired data path, allowing for a high degree of parallelism. However, the price paid for this is that the individual computing units are used only once, and a unit always performs the same operation with the data it receives, i.e., it cannot be used for a different goal. The brain needs a hundred billion neurons (essentially instructions) because practically all elementary instructions need to use another neuron (or better, their combinations, even if a higher level organization unit and its cyclic operation provide re-use possibility). This is why we use only a tiny fraction of our neurons [24] every moment. The brain’s utilization (the payload computing efficiency) is very low. However, its energetic efficiency is high: when not computing, neurons’ maintenance consumes minimum energy [55]. Oppositely, the power consumption of technical computing is enormous.

Although modern processors (mainly the many-core ones) comprise gates comparable in number to the neurons in the brain, the overwhelming majority of those gates only increase the computing speed of a single computing thread, and the cyclic-sequential operating does not need much wiring. We could construct a data-controlled processor from the same number of gates, but it would be a purpose-built system. It would process all data in the same way and would need orders of magnitude times more wiring. Technology cannot imitate the biological mechanism required for the brain's dynamic operation, mainly because of the amount of wiring involved and the lack of materials of unique construction that observe signal propagation time. *We need to bargain between the number of components and wiring on the one side, the architecture's complexity, and the processor's operating speed on the other. Because of this, there are significant differences in the higher-level behavior of technical and biological processors.*

Technical computing is *instruction-flow and clock-controlled*. A complex cyclic-sequential processor *with a fixed external and variable internal data path* is used, which *interprets the data arriving at its input section as an instruction*. After decoding, it sets up its internal state and data paths. After this, the data controls the operation of the processor: until the operation terminates, the processor works in a data-driven mode, and the data flows through the previously set data path.

A sequential processor executes one instruction at a time, considerably limiting the parallelism (although the actual implementations enable some ad-hoc parallelization). The result goes into the output section, and the processor continues in an instruction-controlled regime. In the data-controlled mode, the data can modify the instruction pointer (for example, depending on the sign of the result): the technical processor uses a combination of the two operating modes. In addition, using interrupts implements a kind of event-driven control; it enables multi-tasking. *The temporary instruction-driven mode is needed to make the processor programmable, which is the idea behind the "stored program" concept*. We pay a huge price for the universality: even the relatively simple processors work inefficiently [56].

The decoded instruction must refer to the data to be processed. Instructions would need inaccessibly dense wiring to make their required data directly accessible (by wiring). Furthermore, it would not allow using different data (for example, different vector elements) during processing. For this reason, the instruction references the data through their address in a storage unit, which can be accessed through a particular wiring system (the bus); see Figure 5. However, *the bus can only be used by one computing component at a time*, which means further significant limitations of the parallel operation. Using a bus makes *the arrival time of the data delivered over the bus unknown at design time*. The reconfigurable systems are effective because they use a data path set in advance (instead of setting it after decoding) and use the data paths in parallel. They slow down when they need to access external data. However, it would be possible to use an intermediate (configware) layer [57,58] between the hardware and software layers.

## 2.8. Connecting Elemental Units

Given that real computations can only be implemented as the result of more elementary operation, independently of whether we use instruction or data-driven operating modes, we must connect the chained units and deliver signals from one unit's output section to another unit's input section, even in the case when the same physical unit is used for the consecutive operations (of course, after the corresponding organization and control). Transferring signals needs time (the signal speed cannot be exceeded); furthermore, the information transfer capacity of their connections has theoretical limits [6,7]. One of the fundamental issues when considering the performance of computing systems is that chaining remains out of sight of theorists and technologists. Von Neumann's model correctly included both time contributions. Given that his goal was to establish the operation of their vacuum-tube construction mathematically, he made omissions that, in their case, the data transfer time can be omitted aside from the computing time. He warned that using "too fast processing units" (such as the ones technology produces today) vitiates his famous paradigm and that it was "unsound" to use his simplified paradigm for the timing relations of biological computing. Mainly due to the miniaturization and the increased wiring, the timing relations in modern technological design are firmly approaching the ones that were qualified as "unsound" conditions for applying (the mathematical) computing theory by von Neumann.

### *2.9. Proper Sequencing*

One of the fundamental requirements mentioned by von Neumann is that the instruction must be executed using "proper sequencing". It is a broadly interpreted notion that enables using an instruction pointer, indirect addressing, subroutine calls, callbacks, and interrupts (including operating systems and time-sharing modes). Notice that by using signals "Begin Computing" and "End Computing", we also define the *direction of the time*.

Von Neumann was aware that the temporal length of a computing operation matters (compared to the transfer time defined by science between processing units; the shorter processing time vitiates his omission). His discussion did not consider the that-time already known quantum logic [59], mainly because of its probabilistic outcomes. Quantum processors can solve only particular problems rather than enable the building of general-purpose computers. They run "toy algorithms" with low accuracy (50-90%). "The main challenge in quantum computing remains to find a technology that will scale up to thousands of qubits" [60]. Quantum computing has shown essential advances in the past few years. To build a practically useful computer, one can estimate the number of qubits needed to be 100,000." The technical difficulties make assembling a sufficiently large number of qubits challenging. Using the quantum Fourier transform, one can also imitate classical computing circuits [61]. However, one more principal problem with using quantum computing for building general-purpose processors: synchronization issues ('Begin computing', 'End Computing', 'Operand available' signals, furthermore defining the direction of the time) remain outside the scope.

As discussed in [28,62], one needs three-state systems to describe computing adequately. In the two-state digital electronic logic systems, rising and falling clock edges plus the 'hold' time define time's direction. However, it is questionable if such a third state can be available among the quantum states. "Building such machines is decades away" [60]. This is not only because of technical issues but also because they do not fulfill the "proper sequencing" requirement.

### *2.10. Looping Circuits*

The computing theory assumes simple linear circuits. After making an appropriate organization, one can use some computing units repeatedly, provided that the timing relations are not violated. The simple single-thread processors use the same computing circuitry repeatedly, per instruction. The price paid is that a stored instruction must be fetched and the internal multiplexed machinery reorganized per instruction. Given that the execution time of all elementary instructions must be shorter than a clock period, it does not create additional problems. Notice that using such a solution makes only sense when using instruction-controlled mode: the processor uses an internally changed data path plus instruction-selected external data paths.

## 3. Technical Computing

In his classic paradigm, von Neumann has assumed that the computer will initially be implemented using (the timing relations of) vacuum tubes; furthermore, the computer shall perform a large number of computing operations on a small amount of data and provide a small amount of result. The development of electronic technology quickly outdated the assumed timing relations, and the way computers are used today is far outside their original range of applicability. According to its inventor, *today's processors are implemented with a different(ly timed) technology that makes using the classic paradigm unsound*. Inexpensive computers are used for goals other than computing and are emulated through computing (like synchronization and loop organization, among others). This usage method wastes resources; a processor can connect to the surrounding world using I/O instructions; must imitate (by using memory operations) multi-threading on a single-thread processor; the processor has an inflexible architecture and runs sequential processes. A processor can hardly keep track of and take control of processes occurring in real time.

He made clear that using "too fast" vacuum tubes (or other elementary processing switches) *vitiates his simplified paradigm that forms the basis of today's computing science*: his famous *omission of the* *transfer time from aside the processing time in chained computing operations was valid only for [the timing relations of] vacuum tubes*. In this sense, computing science is the firm theoretical basis for an abstract processor but cannot describe a technically (or biologically) implemented processor. R. P. Feynman, in the late '70s called attention to [63] that *computing belongs to the engineering field and the behavior of a technical processor tends to strongly deviate from the behavior of the abstract processor that the computing science can describe*.

Von Neumann suggested that his proposed simplified computing paradigm should be revisited correspondingly when the implementation technology changes, but this has never been performed. The relevance and plausibility of neglecting the transfer time have never been made questionable; despite that in the brain operation that inspired von Neumann, the non-negligible conduction time (transfer time) was evident from the beginning. However, mathematics, as well as electronic circuit design, and consequently, technological computing stayed at assuming instant interaction, while brain simulation describes the "spatiotemporal" behavior using separated spatial and temporal variables.

We revisited and generalized [21] von Neumann's paradigm, making it technology-independent. The proposed approach correctly interprets the idea of computing in von Neumann's spirit (keeping his correct model). Consequently, it considers that any implementation converts the logical dependence to temporal dependence [19]. Formally, our handling only changes that logical functions (the factual basis of computer science) depend on *where* and *when* they are evaluated. In this way, computer science and the underlying technology are concerted with the modern state of science/technology.

Introducing *time awareness* into (electronic and neurobiological) computing is as revolutionary as in classic science more than a century ago: a series of new scientific disciplines were born. In parallel with science, introducing time into computing science leads to the birth of "modern computing", which can explain the issues that "classic computing" cannot (from increased dissipation of processors through the inefficiency of large-scale computing systems to the nonlinear behavior of algorithms running on them), in complete analogy with as relativistic and quantum physics did several decades ago. Under the current circumstances, the temporal behavior of components, all actors on the computing scene, must be considered in geographically large systems, but inside the miniaturized processor chips, furthermore biological and artificial neurons.

Computing technology is based on elements that have drastically changed during technical development. The precise principles in von Neumann's fundamental work [50], derived from the timing relations of vacuum tubes, are no longer valid for today's technology and materials. However, those out-of-date omissions are behind technical designs.

## *3.1. Cost Function*

Initially, manufacturing transistors was expensive, so the designers minimized the number of transistors in their design. Around 2006, a primary cost function has changed [64]: "Old [conventional wisdom]: Power is free, but transistors are expensive. New [conventional wisdom] is [that] power is expensive, but transistors are free". Today, one can produce a transistor "for the price of sand", and the component's features (memory size, memory cell word length, processor word length, processor data path length, etc.) have enormously enhanced. The need for enhancing design was recognized, but there were no investments to redesign fundamental circuits. "*As we enter the dark silicon era, the benefits from transistor scaling are diminishing, and the current paradigm of processor design significantly falls short of the traditional cadence of performance improvements due to power limitations. These shortcomings can drastically curtail the industry's ability to continuously deliver new capabilities.*" [65]

Technology and circuit design were developed isolated: new designs introduced new "Intellectual Properties" (IP) while keeping the already tested and validated IPs unchanged. Circuits such as an adder were designed to contain a minimum number of gates instead of a path with an identical temporal length for producing results and carry. As we discussed in [21], the increased word length increased the number of unwanted flops, i.e., the non-payload power consumption of processors. Investing in computer processors is a minor portion of the budget; the central part is to provide power (for supercomputers, dedicated power stations) and cooling (river water), so it would be time to change the cost function of building computing systems, including processors.

### 3.1.1. Thermal Limit

Given that only a fragment of the cores (or equivalently, all cores at a fragment of the nominal clock frequency) could be used, and the working cores were operating at their thermal limit, some cores were only used to "pad" the working cores [66,67]. Recently, at high processor densities (such as in the case of supercomputer nodes), the processors' working clock frequency cannot be set to its allowed maximum value. The intense heat production needs intense cooling, which needs space between processors, increasing the data path length and decreasing efficiency and performance. The issues of thermal limit cannot be separated from the issues of increasing the processor's word length, using a central clock, the efficiency of power consumption, and the physical limitations [68].

### 3.1.2. Word Length

As the size of computing tasks grew, so did the address space and the elementary operand's size. The increased operand size and the lack of redesigning the fundamental circuits lead to a drastic power consumption increase. Figure 3 depicts that, due to the wrong cost function (minimizing the number of transistors in the circuit instead of minimizing the time difference between signals' arrival), non-payload flops are present, and their number and ratio exponentially increase with the number of bits in the argument.

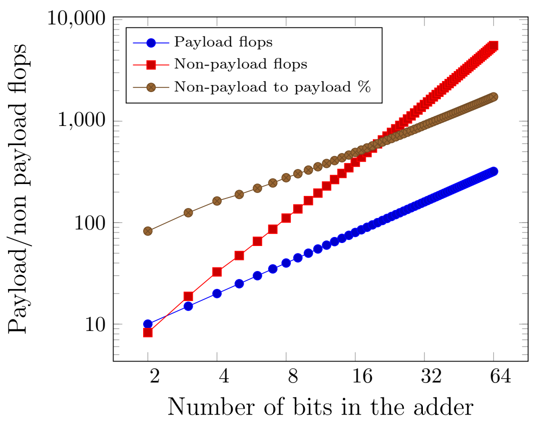

**Figure 3.** Comparing number of payload on nonpayload flops in the function of bits in the argument

### 3.1.3. Wrong Execution Time

Von Neumann clarified that using "too fast" vacuum tubes (or other elementary processing switches) *vitiates his simplified paradigm that forms the basis of today's computing science*. His famous *omission of the transfer time aside from the processing time in chained computing operations* was valid only for [the timing relations of] vacuum tubes. The growing complexity of processor chips needs more dense wiring, increasing the internal data delivery and transfer paths.

From the beginning of computing, the computing paradigm itself, "*the implicit hardware/software contract* [64]", has defined how mathematics-based theory and its science-based implementation must cooperate. Mathematics, however, considers only *logical dependencies* between its operands; it assumes that its operands are *instantly available*. Even publications in most prestigious journals [34,69,70] are missing the need to introduce temporal behavior into computing. The "contract" forces us to develop engineering solutions that imitate a non-temporal behavior. Although the experience shows the opposite, processors are built in the spirit that signals transfer time is neglected aside from its processing time. In a good approximation, that approach is valid for (the timing relations of) vacuum tubes only. Clock domains and clock distribution trees have been introduced to cover the facts. Given that the total execution time comprises data transfer and processing times, it is not reasonable to fabricate smaller components without proportionally decreasing their processing time.

### 3.1.4. Central Clock Signal

Von Neumann suggested using a central clock signal for the initial designs only and at the timing relations of vacuum tubes. His statement is valid for the well-defined dispersionless synaptic delay $\tau$ (where the transfer time can be omitted aside from processing time, and a central synchronization clock is used) he assumed, but not at all for today's processors. The recent activity of considering asynchronous operating modes is motivated by admitting that *the present synchronized operating mode is disadvantageous in the present non-dispersionless world*.

Given that *von Neumann considered the intended vacuum tube implementation*, he proposed neglecting the transfer time, aside from the processing time, *to simplify the model*. Consequently, and *only for vacuum tube implementation (slow processors), he proposed using a central clock signal*. The activity to consider asynchronous operating modes[11,34,44,71] is motivated by admitting that *the present synchronized operating mode is disadvantageous in the non-dispersionless world*. For today's technology (the case of "too fast" processors, as von Neumann coined), the idea of *using a central clock signal cannot be justified*.

### 3.1.5. Dispersion

*The synchronization inherently introduces performance loss*: the processing elements will be idle until the next clock pulse arrives. It was correctly noticed [72] that "*in the current architecture where data moves between the physically separated processor and memory, latency is unavoidable*". In single-processor systems, latency is the primary reason for the low efficiency. The effect grows as the system's physical size grows or the processing time decreases apart from transfer time.

This difference in the arrival times is why von Neumann emphasized: "*The emphasis is on the exclusion of a dispersion*" [50]. Von Neumann's statement is valid for the well-defined *dispersionless synaptic delay* $\tau$ he assumed, but not at all for today's processors and even less for physically larger computing systems. If we consider a 300 m sized computer room and the 3000 vacuum tubes estimated [73], von Neumann considered a distance between vacuum tubes of about 30 cm (1 *ns*) as a critical value. The best case is to transfer a signal between neighboring vacuum tubes. The worst case is to transfer a signal to the other end of the computer room. At this distance, the transfer time is about three orders of magnitude lower than the processing time (between the "steps" he mentioned). With our definition, the dispersion of EDVAC is (at or below) 1 %. We estimate the distance between the processing elements in two ways for modern processors. We calculate the square root of the processor area divided by the number of transistors. This assumption gives the transistors a kind of "average distance" of the transistors, " and we consider it the minimum distance the signals must travel between neighboring transistors. (Notice that this transfer time also shows a drastic increase with the number of transistors, but alone does not vitiate the classic paradigm). This value is depicted as "Proc transfer" in Figure 4. The maximum distance between the two farthest processing elements on the chip is the square root of the processor area. Introducing clock domains and multi-core processors evidently shades the picture. However, we cannot estimate more accurately without proprietary technological data.

As the "Proc dispersion" diagram line shows, *the dispersion is near unity in today's technology. We can no longer apply the "dispersionless" classic paradigm.* Reaching the plateau of the diagram lines coincides with introducing the "explicitly parallel instruction set computer" [74]: the maximum that the classic paradigm enabled. On one side, it means that *today, electric power is almost entirely wasted because of the wrong theoretical base*. On the other, this means that the operating regime of today's processors is closer to the operating regime of the brain (where explicit "spatiotemporal" behavior is considered and it is "unsound" to use the classic paradigm to describe it) than the operating principle abstracted in the classic paradigm.

Von Neumann used the word "dispersion" only in a broad (and only mathematical) sense, so we quantitatively define merit for dispersion (the idea can be concluded from his famous fundamental report) using technical data from the given implementation. We provide a "best-case" and a "worst-case" estimated value for the transfer time and define the dispersion as the geometric mean of the minimum and maximum "Proc transfer" times, divided by the processing time. The rough idea should be refined, and the dispersion shall be derived from proprietary technological and topological data.

Given that processing elements and storage elements usually are fabricated as separated technological blocks (this technical solution is misinterpreted as "the von Neumann architecture"; the *structure* and *architecture* are confused), and they are connected by wires (aka bus), we also estimated a "bus transfer" time. The memory access in this way is extended by the bus transfer time (this situation motivates the efforts to make "in-memory computing"). Because of this effect, we assumed that cache memory is positioned at a(n average) distance of half processor size. This time is shown as "Cache transfer" time. The cache memories appeared about the end of the 1980s when it became evident that the bus transfer non-proportionally increases the memory transfer time (cache transfer time data can be calculated for processors without cache memory, however).

Given that EDVAC and Intel 8008 have the same number of processing elements, their relative processor and cache transfer times are in the same order of magnitude. However, the importance of bus transfer time has grown and started to dominate single-processor performance in personal computers. A decade later, the physical size of the bus necessitated introducing cache memories. The physical size led to saturation in all relative transfer times (the real cause of the "end of the Moore age" is that Moore's observation is not valid for the bus's physical size). The slight decrease in the dispersion in the past years can probably be attributed to the sensitivity of our calculation method to the spread of multicores, suggesting repeating our analysis with proprietary technological data.

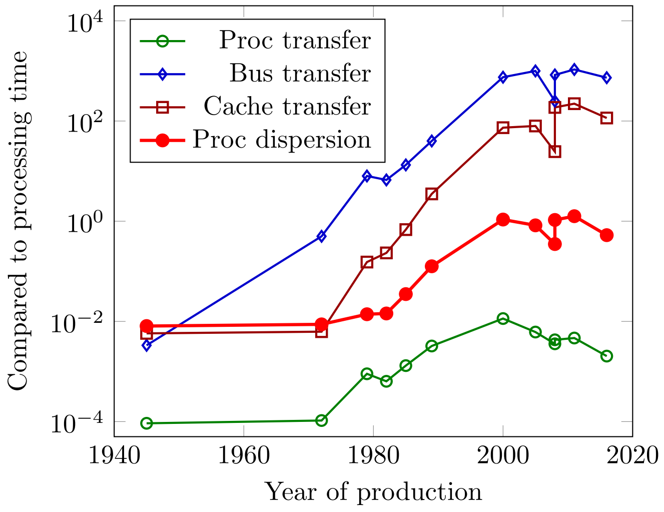

**Figure 4.** The history of some relative temporal characteristics (dispersion) of processors, in function of their year of production. Notice how cramming more transistors in a processor changed disadvantageusly their temporal characterisctics.

### 3.1.6. Generating Square Waves

Digital technology needs square waves. The sooner a signal reaches (90% of) its amplitude, the better. However, more energy is needed to do it. Action Potentials resemble a damped oscillator with $\zeta = 0.3$. The damping means not only speed but also energy consumption. Being digital also means losing the option of having analog memory due to damping.

### 3.1.7. Resource Utilization

The internal circuits for performing elementary operations can be subdivided into sections (performing "more elementary" operations) that provide inputs/outputs for each other. In this way, they can be organized to work simultaneously on executing different instructions. Pipelining needs interfacing with those sub-instructions, considerably increasing the need for data moving and temporary storing; it needs extra organization. Furthermore, they were designed for sequential operation, and non-sequential operations (such as interrupts, multithreading, and conditional jumps) kill the pipeline. The idea roots back to the times when "transistors were expensive".

To increase single-processor performance, an apparently good idea is to use multiple-issue computing sub-units (one can classify designs such as GPU, multiple arithmetic units, conditional jump, and out-of-order executions). However, they cannot be utilized at their full power. The benchmarks show genuine results when all issues have operands with which to work. It is the end-user's responsibility to organize the workload to maximize the available computing power; if possible.

## 3.2. Hardware/Software Cooperation

Computing was, for decades, a sound success story with exponential growth. The provided computing performance was aligned with the computing demand for a long time. About two decades ago, however, as the report of CSTB of the US National Academy of Sciences stated, "*Growth in single-processor performance has stalled – or at best is being increased only marginally over time*" [75]. Experience showed that wiring (and its related transfer time) tended to have an increasing weight [68] in the timing budget, even within the core. When reaching the technology limit of around 200 *nm* wiring started to dominate (compare this date to the year when the dispersion diagram line reached its saturation level in Figure 4). Further miniaturization can enhance the computing performance marginally and increase the number of issues due to approaching the limiting interaction speed. Since the computing industry heavily influences virtually all industrial segments, stalling crashed long-term predictions in industry, military, research, economy, etc. The shortage of integrated circuits first became visible in the early 20's, and recently, the proliferation of the AI systems causes a shortage in computing power itself. The demand for computing keeps growing and shows an unsustainable increase, see Fig. 1 in [35]. The increase in computing power is far from linear and divides into "ages", marked by the beginning of using some technology. Until 2012, computing power demand doubled every 24 months; recently, this has shortened to approximately every two months. Mainly due to the wide proliferation of AI, the tendency keeps continuing.

Computing, commonly called "computing stack", is implemented in really many layers with many interfaces between. Executing software instructions is using electronic gates in another way. We also introduce different layers within the software layer, such as application, OS, networking, and buffers with their separate handling. Although using interfaces between them simplifies thinking and makes developing less expensive, it decays efficiency, mainly due to the need for moving data.

### 3.2.1. Single-Thread View

In the original idea, safety and comfort were not present, so it remained for the software layer to provide it, although hardware created some tools. The operating system attempts to map the abstract mathematical operation to hardware. On the one side, it provides the illusion for the hardware that only one process runs on it and for the software that it has exclusive access to the hardware. However, at critical points, the hardware disappears for all software processes, and 'foreign' instructions are executed by the OS. It was early noticed that [76] "operating systems are not getting as fast as hardware". Too much functionality remain for the SW. Even typical HW tasks such as priority handling [77,78] remain for the SW.

The elementary size of executable instructions is too small. When several different tasks (and threads) share the processor, the change between them and the access to the processor's critical resources need a forceful intervention (context switching), which is very expensive regarding executable instructions. To implement it for using the functionality of the OS, executing an I/O instruction or sharing resources (mutex), about 20,000 machine instructions must be executed [41,42].

Although G.M.Amdahl in 1968 question marked [79] the "validity of the single processor approach to achieving large scale computing capabilities", even today, the processors are optimized (and benchmarked) for single thread processing, they are used under OSs and in many-thread environments. When switching threads (and even when calling subroutines), logically, a new processor is needed, so the OS or the application must save and restore (at least part of) its many registers using time-expensive memory operations. The instruction sets working with 'registers only' instructions attempt to reduce unneeded memory access. Using the 'processors are free' idea and a technical solution similar to the "big.LITTLE technology" [80], an alternative mode of using many-core processors, could help a lot [57,58]. As was demonstrated [81], cooperating processors significantly increase the efficiency of heavily communicating systems.

Unfortunately, the original paradigm did not theorize making I/O operations and, during the times, those operations were implemented as accessible through the OS. In the age of 'Big Data', 'Internet of Things' and graphical OSs, making I/O operations can represent a significant share of the executed instructions, given that their number must be weighted with the needed context changes.

### 3.2.2. Communication

Attempting to construct an artificial system of neurons with at least approximately as good features as the nature constructed is hopeless. "In the terminology of communication theory and information theory, [a neuron] is a multiaccess, partially degraded broadcast channel that performs computations on data received at thousands of input terminals and transmits information to thousands of output terminals employing a time-continuous version of pulse position. Moreover, [a neuron] engages in an extreme form of network coding; *it does not store or forward the information it receives but rather fastidiously computes a certain functional of the union of all its input spike trains,* which it then conveys to a multiplicity of select recipients" [46]. "A single neuron may receive inputs from up to 15,000-20,000 neurons and may transmit a signal to 40,000-60,000 other neurons" [82].

### 3.2.3. Wiring

Computing science has established itself on von Neumann's simplified paradigm and instead of working out mathematical formalism for the modern (and future) technological implementations, quietly introduced an "empirical efficiency" for different computing systems, instead of understanding theoretically that computing inherently comprises efficiency and many of the shocking experiences, such as that the absolute performance decreases when increasing the number of processors in parallel processing systems [83][84], that the efficiency of a computing system depends on its workload [54], that the number of articial neurons in an ANN that contribute to the resulting performance is limited [85] [86], that a vast system needing heavy communication can collapse [15], that the need for communication restricts the *usable HW* to the 1% of the *available HW [14].*

Unexpectedly, that omission has far-reaching consequences and infects the thinking of HW designers. Claims such as "supercomputers, which, unlike the brain, physically separate core memory and processing units. This slows them down and substantially increases their energy consumption" are available. Furthermore, "The brain as a physical substrate of computing is also fundamentally different from general-purpose computers based on digital circuits. It is based on biological entities such as synapses and neurons instead of memory blocks and transistors." [34] Neither sentence mentions the role of connection needed between those units. However, although the data transmission time is about two orders of magnitude higher than the processing time in biology and is getting similar to that in technical systems as the system size grows. The slow-down is not the consequence of separating memory and processor; it is the direct consequence of omitting the data transmission time. Also, the experienced nonlinearity originates from the fact that the total execution time comprises a sum of the operating and transferring time, and the proportion of the latter, omitted time, increases, and the technology proceeds.

The chain's first operating unit receives the input data in both modes, and the last unit provides the result. Many elementary computing objects shall be provided for more sophisticated and lengthy computations. Alternatively, after making an appropriate organization, some units can be re-used. The first control option means wasting computing components. While it enables a considerable parallelization for short instruction sequences, its operation needs nearly no organization. The second option requires an organizer unit. While it enables minimum parallelization (presumes that the instructions use some computing units exclusively), it allows unlimited operating time and many computing operations. Anyhow, *both control modes need a timing alignment for the correct operation.*

Data-driven operating mode allows for a high degree of parallelism, given that *all data have their exclusively used data path*. The price paid is that the individual computing units are used only once, and a unit always performs the same operation with the data it receives. The brain needs a hundred billion neurons (essentially *instructions*) because practically all elementary operations need to use another neuron (even if a higher level organization unit provides re-use possibility). This feature enables us to use only a tiny fraction of our neurons [24] every moment.

Although the neurons in the brain are comparable in number to the gates in modern processors (mainly the many-core ones), the overwhelming majority of those gates only increase the computing speed of a single computing thread, and the cyclic-sequential operating does not need much wiring. We could construct a data-controlled processor from the same number of gates, but it would be a purpose-built system. It would process all data similarly and need orders of magnitude times more wiring (see, in some sense, attempts to build AI systems with over a hundred billion [87]). The needed amount of wiring exceeds technological possibilities. Given that the paradigm does not consider the signal propagation time, technology cannot imitate the biological mechanism required for the brain's dynamic operation. *We must bargain between the wiring and the number of components on the one side, the processor's operating speed, and the architecture's complexity on the other. Because of this, there are significant differences between the elementary operations and the higher-level behavior of technical and biological processors.*

Technical computing is *fundamentally instruction-flow and clock-controlled*. It uses a complex cyclic-sequential processor with a fixed external data path, which *interprets the data arriving at its input section as an instruction*. It sets up its internal state and data paths after decoding. Following that, the data controls the processor's operation. Until the operation terminates, the processor works in a *data-driven mode*, and the data flows through the previously set data path.

Although the actual implementations enable some ad-hoc parallelization, a sequential processor executes *one instruction at a time* in the data-controlled mode, considerably limiting the parallelism. The result goes into the output section, and the processor continues in an instruction-controlled regime. In the data-controlled mode, the data (for example, when the result is negative) can modify the instruction pointer: *the technical processor uses a combination of the two operating modes*. Using interrupts implements a kind of *event-driven control*; it enables multi-tasking. *The idea behind the "stored program" concept is that the temporary instruction-driven mode is needed to make the processor programmable*. This mode enables single-thread, sequential (von Neumann-style [19]) programming, and implementing the functionality missing from the paradigm cannot keep pace with the hardware processing speed [76]. The huge price paid for the universality: even the relatively simple processors work inefficiently [56].

The decoded instruction must find its data to be processed by referring to them. To make their required data directly accessible (by wiring), the instructions would need inaccessibly dense wiring. Furthermore, it would not allow for using different data (for example, different vector elements) during processing. For this reason, the data can be accessed through a particular wiring system (the bus), and the instructions reference the data through their address in a storage unit. However, only one computing component can use the bus at a time, which means further significant limitations of the parallel operation: the software threads must compete (through the operating system) for accessing their instructions, and the instructions must compete for the data they need. What is worse, *using a bus causes the time of arrival of the data delivered over the bus to be unknown at design time*. The reconfigurable systems are apparently effective because they use a data path set in advance (instead of setting it after decoding) and enable using data paths in parallel. However, it would be possible to use an intermediate (configware) layer [57,58] between the hardware and software layers, with cooperating processors and a hierarchically organized distributed bus system.

### *3.3. Structure Vs. Architecture*

In his fundamental work, von Neumann clarified that the document was about logical organization. The technology formed technology blocks and introduced (based on the idea that the transfer time was negligible) a universal wiring system that connected the blocks. It was correctly noticed [72] that "*in the current architecture where data moves between the physically separated processor and memory, latency is unavoidable*". We add that to move complex processors into distributed memories or globally accessible memory into processors is also impossible; this is the price paid for using instruction-driven mode.

### 3.3.1. Single-Processor Performance

A decade ago, the limits of computing, the laws of nature enabled [68], had already been reached. Processors are optimized for single-thread performance [74]. One of the limitations (which indeed caused a saturation of single-thread performance around 2005, was that simply no more reasonable functionality could be added to recent processors: "The exponential growth of sequential processors has come to an end" [88], precisely as it was contented a half-century ago: "the organization of a single computer has reached its limits" [79]. One of the many other reasons is that Moore's observation is valid only for the density of components inside the processor. The rest of the computing systems, including their *buses connecting the components, do not decrease at a similar rate*. This circumstance should result in an S-curve [89] rather than an exponential increase. This case is valid in human-size computing systems and inside processors. Consequently, the relative weight of data transfer, as well as of other non-payload activities, increases with time. Rather than rethinking computing, preparations for the post-Moore era [90] were initiated.

The expectations against computing are rather demanding and excessive. Billions of processors are present in the edge devices. Millions of processors are crammed into single large-scale computing systems. Computing targets extreme and demanding goals, such as global weather simulation, Earth simulation, or brain simulation. These tendencies, accompanied by the growing amount of "Big Data", "real-time connected everything", and virtually infinite need for details of simulations, not to mention the irrationally enormous AI-purpose systems [87,91], are a real challenge. In several fields, it has already been admitted that the limitations of computing are near or already reached. Today, a growing number of demonstrative failures (both hardware and software) showed severe deficiencies in the fundamental understanding of computing. "Rebooting computing" [52,53] and "reinventing electronics" [92] are a real need, but no valuable ideas have been published, especially *no appropriate computing paradigm*.

### 3.3.2. Multi- and Many-Core Processors

After the single-processor performance stalled, the multi- and many-core processors could provide for a short period the needed higher performance, without a dramatic performance loss, at the price of higher power consumption and lower efficiency. The developed GPUs were well usable for their original goal. However, as general-purpose [93] accelerators, they introduced nearly two orders of magnitude worse computing efficiency and about an order of magnitude worse power efficiency. Similar is the case with the other SIMD processors. They enhance much when used in the scenario they were designed for but significantly less in general cases of use [20].

Until GPU appeared, the need for computing power and its supply was moderate, and they roughly followed Moore's observation. With the appearance of GPU, the need for computing power suddenly (in a leap-like way) increased. With the appearance of AI (actually, mainly ML), the increase became steeper, and the same happened (not yet shown in the figure) with the appearance of the LLM-based generative AI systems. The overhyped popularity of using ChatGPT and others made it evident that the development of computing ran into a dead-end street [94], primarily because of using inappropriate computing theory, causing sustainability issues with its power consumption, carbon emission, and environmental pollution. It is time to revisit computing (i.e., making drastic changes, starting from the basic principles and scrutinizing theoretical assumptions and omissions, operating principles, etc.).

### 3.3.3. Memory

In technical computing, *registers are essentially another memory* with short addresses and short access times; simulating data stored directly at synapses would require many registers. *Using conventional "far" memory instead of the direct register memory introduces a worse performance of two to three orders of magnitude. In machine learning, summing synaptic inputs means scanning all registers, which requires several thousands of machine instructions and introduces another three to four orders of magnitude worse performance.* In addition, the correction term, calculated from the gradient, is distributed between all upstream neurons, whether they affected or not the result; it decreases again about two to three orders of magnitude.

### 3.3.4. Bus

The theoretical schematics of neural networks are the exact equivalent of biological connectivity; see Figure 5, left side. However, the technologically implemented data transfer needs a bus, with all its disadvantages. The first node forwards its data to the bus, and the second receives it from the bus, right side. The principle of the operation (computing science) matches the left side of the figure, and the practical operation (computer science) differs significantly (see the right side of the figure), also implying significant differences in their features. The bus can only be used with exclusive rights, so all neurons must contend for the right to use the bus (arbitration) in all cases when they want to use it. Bus arbitration contributes a (long) pseudo-random time to the individual transfer times and introduces considerable uncertainty into the design. Depending on the system's size, that contribution may become more dominant.

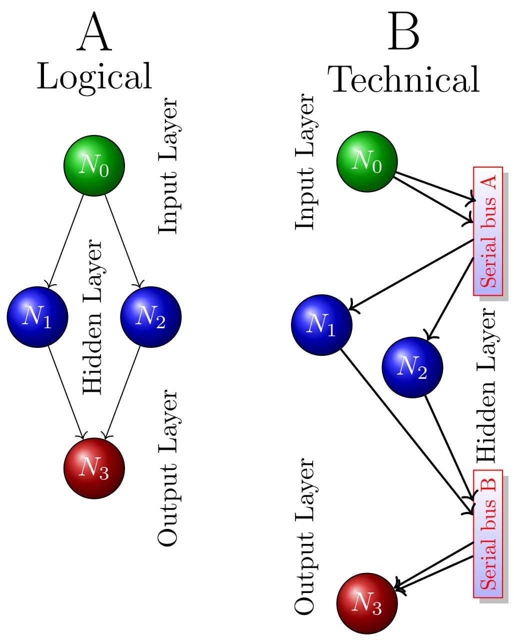

**Figure 5.** The neuronal information transfer data paths. A: Data path of the logical and biological transfer. B: Data path of the technical transfer

In biology (and mathematics), the data transferred in parallel are serialized in the technical implementation. The bus can transfer only one piece of data at a time. If many neurons are connected to the bus and want to transfer their data simultaneously, the bus bandwidth may significantly limit the transfer time. *Using a bus makes the width of the transfer's time window indefinite*; the larger the system, the more indefinite it is.

The bus is not the only component responsible for the inefficiency of computing stacks. In addition to using the bus intensively, I/O operations also need the assistance of the operating system in the form of "context switching" that consumes about $10^4$ machine intructions [41,42] per operation. It was bitterly admitted that: "*artificial intelligence, . . . it's the most disruptive workload from an I/O pattern perspective*". [43]. *von Neumann's simplified paradigm did not plan I/O: his computing paradigm was about performing a large amount of computation using a small amount of data in the memory* (except reading input data at the beginning and printing results at the end).

From a computational point of view, biological axons represent a private connection between computing units, but billions of such (shallow speed) "buses" work in parallel without contention. The usual technological implementation is using a single, high-speed bus, which must be "owned" for every communication. Using a serial bus means spending most of the processing time with arbitration. Given that the needed arbitration limits the transfer time (increases with the communication intensity instead of the number of neurons!) rather than by the bus speed, using a single high-speed bus is at least questionable in large-scale systems: "The idea of using the popular shared bus to implement the communication medium is no longer acceptable, mainly due to its high contention." [95] Bus arbitration, addressing, and latency prolong transfer time (and decrease the system's efficacy). This type of communicational burst may easily lead to a "communicational collapse" [15], but it may also produce unintentional "neuronal avalanches" [96]. As discussed in [97], the shared medium becomes the dominant contributor to computing's time consumption at many communicating units. Given that conventional processors are implemented in SPA, as Amdahl coined the wording [79], communication between them is implemented through I/O instructions. We can hope for better performance only in a drastically different approach [98].

A sequential bus can deliver messages only one after the other, i.e., *increasing the number of neurons increases utilization of the bus and prolongs the total execution time as well as apparent processing time of the individual neurons*. The effect can be so strong in large systems, that emergency measures must have been introduced: the events "*are processed as they come in and are dropped if the receiving process is busy over several delivery cycles*" [14].

Figure 6 shows the actual timing of a bus transfer from two competing components. Two neurons at two different (temporal) distances from the bus want to send their data simultaneously. The payload time (transferring data on the bus, represented as vertical green arrows) dwarfs aside from the summed values of bus request (BRQ), bus grant (BGT), delivery to/from the bus times (Bdt). A bus with a higher speed would not help much.

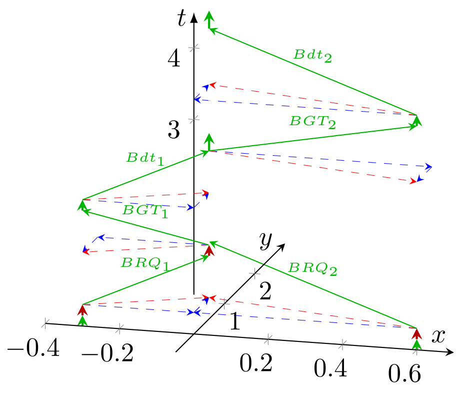

**Figure 6.** Timing relations of a bus transfer for two competing components

Figure 5 shows the "technological implementation" of the neural-imitating communication over a sequential bus; the timing relations are those shown in Figure 6. The inset shows a simple neuromorphic use case: one input and one output neuron communicating through a hidden layer comprising only two neurons. Figure 8A mostly shows *the biological implementation: all neurons are directly wired to their partners*, i.e., a system of "parallel buses" (the axons) exists. Notice that the operating time also comprises two non-payload times ($T_t$): the data input and output, which coincide with the non-payload time of the other communication party. The diagram displays logical and temporal dependencies of neuronal functionality. Payload operation ("the computing") can only start after the operand is delivered (by the, from this point of view, non-payload functionality: input-side communication), and output communication can only begin when the computing is finished. Recall, that, importantly, communication and calculation mutually block each other. Two important points that neuromorphic systems must mimic are noticed immediately: i/ *the communication time is an integral part of the total execution time*, and ii/ *the ability to communicate is a native functionality* of the system. In such a parallel implementation, the system's performance, measured as the resulting total time (processing + transmitting), *scales linearly with increasing non-payload communication and payload processing speeds*.

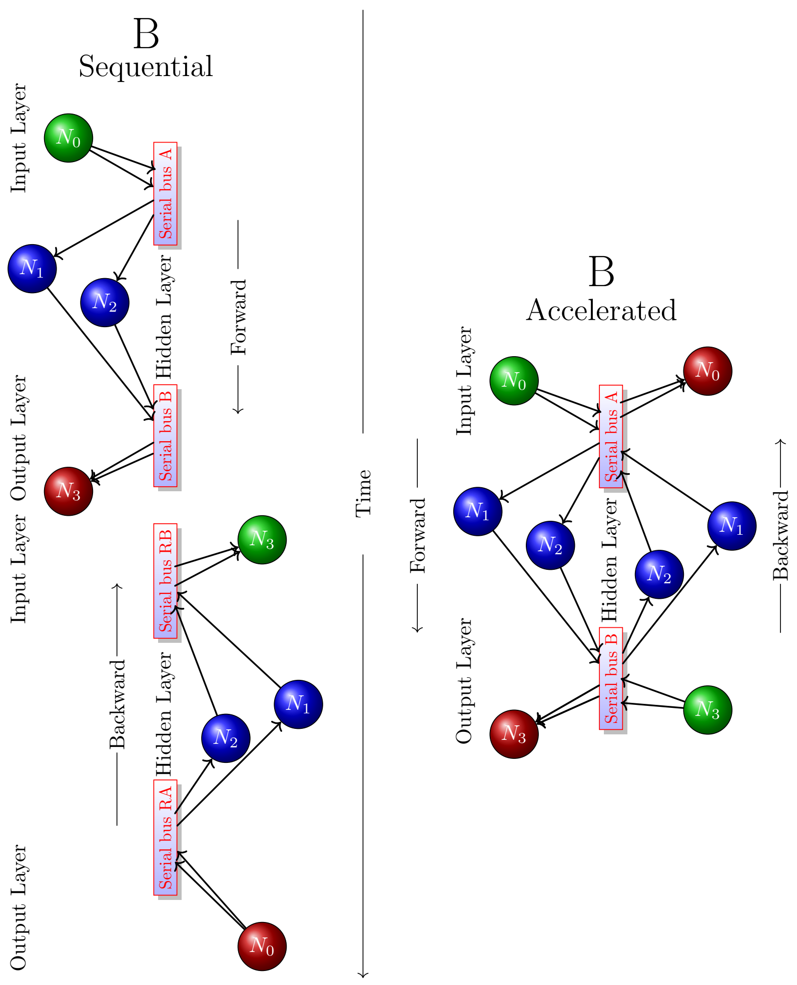

**Figure 7.** The information transfer data paths in neuronal deep learning. A: Data path of the sequential layer operation transfer. B: Data path of the parallel layer operation transfer

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 February 2025 doi:10.20944/preprints202502.2238.v1

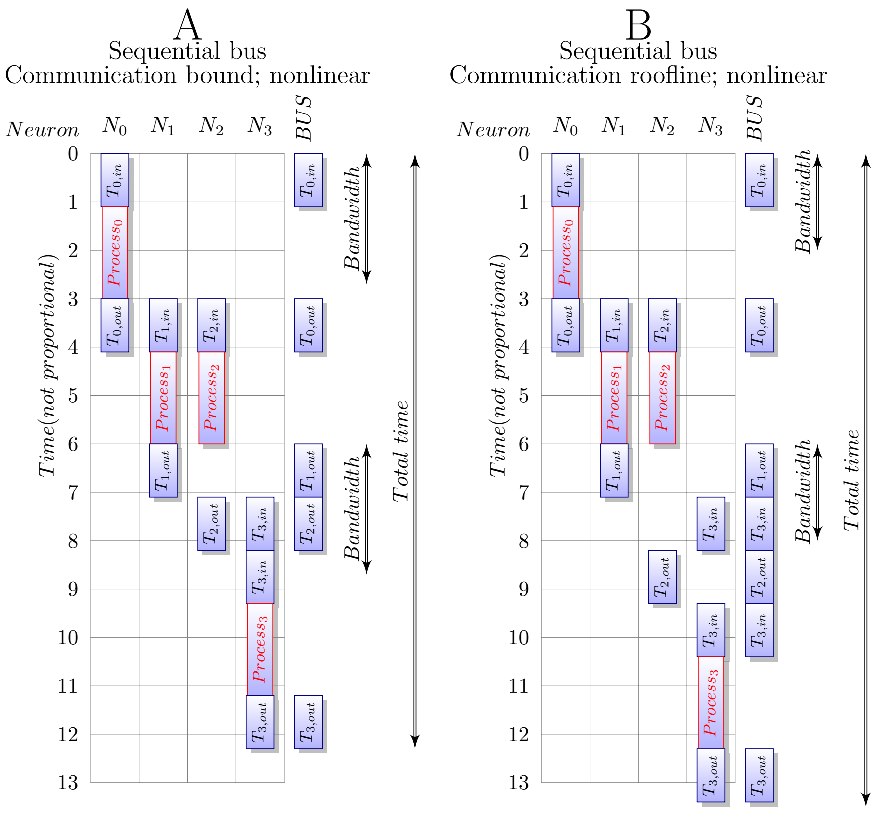

**Figure 8.** "Roofline" limiting effect of the serial bus

Figure 8B shows a *technological implementation of a high-speed shared bus* for communication. The activity that loads the bus at the given time is shown to the right of the grid. A double arrow illustrates the communication bandwidth, the length of which is proportional to the number of packages the bus can deliver in a given time unit. We assume the input neuron can send its information in a single message to the hidden layer. Furthermore, processing by neurons in the hidden layer starts and ends simultaneously. However, neurons must compete to access the bus, and only one can send its message immediately; the other(s) must wait until the bus gets released. The output neuron can only receive a message when the first neuron completes it.

Furthermore, the output neuron must first acquire the second message from the bus, and processing can only begin after having both input operands. *This constraint results in sequential bus delays during non-payload processing in the hidden layer and payload processing in the output neuron*. Adding one more neuron to the hidden layer introduces one more delay period.

## 3.4. Accelerating Computing

We assume that a computing machinery (with a "black box" operation) exists that starts having its needed operand(s) in its "input section(s)" and terminates having the result(s) in its "output section(s)". Even if the same computing unit physically provides the input and output sections, data transfer must deliver capacity, control, and time. The adjacent operations are implemented by data transfer between computing units. To speed up the operation, at the price of using more resources (including technology blocks and energy), one can use "multiple issue" operations for calculation and processing. However, the initial research discovered that "code transformations designed for multiple-instruction-issue processors are found to be effective for all register file sizes; however, for small register files, the performance improvement is limited due to the excessive spill code introduced by the transformations” [99]. Unfortunately, the statement is valid for all kinds of using accelerated machinery: the “spill code” matters.

The machinery can only perform elemental operations, so another delivers data to and from its corresponding sections. We broadly interpret the notion of "computing" to apply to elementary state-changing operations of transistors and neurons, composite circuits of logic gates and neuronal assemblies, sharing tasks by computing threads and regions of the brains, and even to the cooperation of technical or living entities. Notice that we explicitly assume that data transfer is needed to deliver data; in technical systems, from elementary circuits to processors to networks, in biological systems, from neurons to brain. Furthermore, any of the above operations can be made ’more intense’ by parallelizing (performing simultaneously) more than one resemblant operation. Depending on the logical sequencing, computing operations can split and join. We have two crucial aspects: how fast, globally, the operation is (i.e., what time is required to perform the needed computation) and what is the cost of the operation (with its many aspects, from the carbon and water footprint of manufacturing the machinery to energy consumption to financing). The time is absolute, but the demand for computing capacity is exploding. The cost function depends on many factors and is enormously affected by materials, technology, physical effects. Both of them are heavily affected by computing science, the latter in a broad sense, again: not only the algorithms but also the theory of computing. Apparently, making some activities simultaneously increases performance, but the “spill code”evidently introduces extra activity that needs extra resources (including power) and time, so it, per definition, decreases computing efficiency.

We quantitize the efficacy of computing machinery based on how much time they spend with the raw computing operation and the supplementary operations such as delivering data to the machinery, preparing the calculation, and waiting for the arrival of data. The discussion recovers that we can separate the computing into stages when we subdivide the total time of operation into a “payload” operation when the computing machinery prepared operands are processed and “non-payload” operations when the “payload” operations are serviced. Notice that only the first item contributes to the nominator in calculating the efficacy, although it cannot live without the second item. *The inefficiency is built into computing; the question is how to control their ratio.* The input operands must arrive at the processing element before the operation begins, and the delivery of the output operand can begin once the computing process terminates. Furthermore, the components must be notified that their needed operands are *available*. The requirements are always valid for geared wheels, biological, neuromorphic, quantum, digital, analog, neuronal, and accelerated processing.

Computing systems have always been expensive toys. The goal was to decrease expenses while increasing computing performance. We need a cost function to measure how far the goal is, and we set up the implementation technology according to that function. Today, cost functions have changed. In many cases, the critical aspects are the absolute execution time, the absolute power consumption, the needed absolute investment to build a new system, the energy cost per computing operation, etc.

It was and is hard to bargain between those conflicting points of speed and expenses. In many cases, the design principles needed to catch up with the quickly changing technology. As discussed above, introducing an instruction-driven mode led to substantial performance losses. Different implementation methods introduce further losses. This section shows examples of how temporal analysis enables one to estimate performance in advance or find mistakes in the existing implementation.

### 3.4.1. ’Multiple Data’ Computing

GPUs have been invented to accelerate graphic processing, which was appropriate for that goal. However, misunderstanding their operating mode made their general purpose use increasingly widespread. It is a fallacy that they can produce about two orders of magnitude higher *payload* computing performance (valid only to their nominal performance) than the CPU alone. In reality, they can make two to three times higher *payload* performance [93] while needing an order of magnitude more electric power.

The GPU performs the same operation on multiple data (say, making the same transformation on all points of the screen or making the same 'multiply and add' operation for all synaptic inputs of a neuron; it is ideal for neural computations). It makes the instruction fetch and decode operations only once, and the sufficiently broad datapath enables operating on many data values. However, GPU makes the interpretation of its time windows doubtful. The input data must be transferred to its input section, and after performing the computation, the results must be transferred to its output section. That is, the arrival/processing/delivery of data happens gradually and partly simultaneously. The operation time can be as short as one single clock period if the unit waits for the arrival of all arguments. However, in that case, the data transfer time is very long since delivering data to and from the unit takes one clock period per data (the memory cells shall be read/written individually, even if it is done in some hidden way). However, one can operate with the incoming data immediately, too: GPU's sophisticated stalling/interleaving mechanism ensures that it will process all data at some time.

We could interpret the *processing time window* approximately as the time from the beginning of the first computation the GPU starts to the termination of the last computation it performed, given that at the edges; the window overlaps with the corresponding *transfer time window*. *The time windows of data transfer and data processing (depending on the data traffic) may overlap, not fulfilling the requirement of Neumann's computing paradigm.*

*The time window's width is unknown, but it must be fixed at design time*. It must not be too large since it decreases operation performance. Suppose it is designed with only a moderate reserve time, and something causes a considerable delay during the operating time. In that case, part of the data arrives outside the time window, which is an actual danger for input and output data.

A GPU unit loads and reads its memory simultaneously while operating. It works with memory cells and operates with the cells, whether or not they contain the intended argument. If the bus does not deliver in the planned time (the data arrives outside the time window), GPU works with "foreign" data. One cannot notice the issue in the case of video applications, where GPU transfers unbiased data only in one direction over one bus. An occasional error manifests that one of the million pixels has a wrong value for a 10 msec period. However, *in deep learning, data from many GPUs depend on each other and are intermixed on a single bus*. Due to the arbitration, they delay each other. In addition, results are used for computations that persist in the synaptic weight for some time.

To avoid that effect, one needs to wait for the arrival of all neural inputs (asynchronous computing) or to go ahead in the absence of the missing value (i.e., hoping that the correct value would not differ very much from the actual content of the corresponding memory) or to repeat the computation upon arrival of a new input (inducing the same effect in the deeper layers). The middle case introduces an undefined component into the result, and the third option repeats the time window, mainly increasing the power consumption and heat production, furthermore decreasing computing efficiency.

### 3.4.2. New Materials/Technologies for Data Storing

Memory access became the "von Neumann bottleneck" of computing [21]. The need to use different storage elements, various costs, storage capacities, and access times was known in computing from the beginning. They started to appear in practical computing systems as cache memories at different levels when memory access began to limit the time of solving computing tasks with the increasing processing speed. The experience has shown that computing speed is sensitively dependent on caching, but the theory does not admit that it is the physical access time that limits performance. On the one side, caching vastly increases the need for data moving (and, consequently, the power consumption of handling stored data).

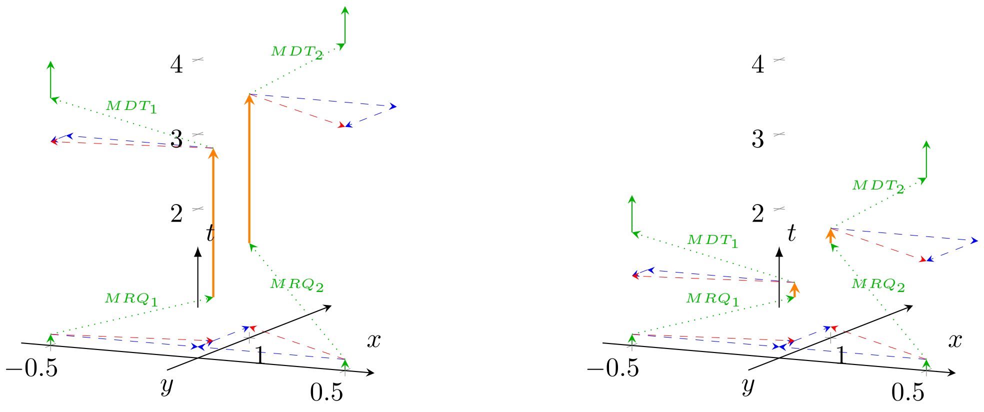

**Figure 9.** Twodifferent cache memories, with the same physical cache access speed, but at different internal on-chip cache position. Cores at x=-0.5 and x=0.5 positions access on-chip cache at y=0.5 and y=1.0, respectively. Vertical orange arrows represent physical cache operating time, and vertical green arrows the apparent access time. Assumes new material/physical mechanism.Left side: Normal speed cache memory. Right side: Super-quick (10 times quicker) cache memory.

On the other hand, it has been forgotten that *in applications of a type such as 'Big Data' and 'Machine Learning', the principle of 'data locality' can be applied less and less*. On the third side, the enormous physical size (machines of the size of several tennis courts) and amount of data (using several billions of weights in the newest "brute force" systems) called attention to the needed power consumption and the physical delivery time, especially over arbitrated buses. The apparent processing time is only slightly affected by the physical speed of the cache memory. As we show examples in [21], if some new material/technology/effect (at considerable expenses) could decrease cache physical access time to one-tenth of the time of the old technology/material, the apparent access speed could only be improved by a factor of less than two. Even if one could reduce the physical cache time to zero, the apparent access time cannot be reduced below the time defined by the respective distances/interaction times. It is insufficient to produce cache memory with faster physical operation; it must be positioned closer to the PUs to speed up the computing process: *all contributions to the operating times must be proportionally reduced to achieve a quasi-linear speedup*.

### 3.4.3. Using Memristors for Processing

Similarly, a frequently proposed idea is to replace slow digital processing with quick analog processing [100,101], and it may be suggested that any new physical effect and material be used in the future. Supposing the memristor is not placed inside the PU, the timing relation remains valid for this improvement. Inserting the memristor into the pipeline leads to different problems, such as creating a connection with the digital processing that cannot directly be connected to a unit making analog processing. One can follow a similar analysis concerning new physical effects in [53]. They decrease the processing time, but to make them useful for computing, their in-component transmission time, especially inter-component transmission time, must be considerably reduced.

It sounds good that "The analog memristor array is effectively the neural network laid out in the form of a crossbar, which can perform the entire operation in one clock cycle" [44]. In brackets, however, reasonably added that "(not counting the clock cycles that may be required to fetch and store the input and output data)". All operands must be transferred to its input section (previously, they must be computed or otherwise produced). Furthermore, the results must be moved from their output section to their destination; see the general computing model in Figure 1. In a fair comparison, one shall compare the effective computing time of computing operations, from the beginning to the end, of memristor-related and conventional operations. (The problem persists even if continuous-time data representation [102] is used. The two orders of magnitude enhancement in efficiency in [103] has arisen from lacking the serial bus connection rather than using memristors.) *The data delivery time must not be omitted.*

#### 3.4.4. Using Mixed-Length Operands

Another common fallacy is that using shorter operands decreases computing time proportionally. It is valid for power consumption and processing speed, but the data transmission time remains practically the same; for a detailed analysis, see [54].

The so-called HPL-AI benchmark used Mixed Precision rather than Double Precision computations. The name suggests that AI applications may run on supercomputers with that efficiency. However, the type of workload remains the same, so that we can expect the same efficacy as double-precision operands. For AI applications, the efficiency can be better by a factor of up to $2 \cdots 3$, depending on the workload and size of the system. However, the enhancement comes from accessing fewer data in memory and using faster operations on shorter operands instead of reducing communication intensity.

It is a common fallacy that benchmark HPL-AI is a benchmark for AI systems. It means "The High-Performance LINPACK for *Accelerator Introspection*" (HPL-AI), and "that benchmark seeks to highlight the convergence of High-Performance Computing (HPC) and AI workloads", see https://www.icl.utk.edu/hpl-ai/. It uses the operand length commonly used in AI tasks but does not have much to do with AI. However, even https://www.top500.org/lists/top500/2020/06/ mismatches operand length and workload: "In single or further reduced precision, which are often used in Machine Learning and Artificial Intelligence applications, Fugaku's peak performance is over 1,000 petaflops (1 exaflop)".

### *3.5. Mitigating Communication*

Different methods for "pruning" [104] serve to reduce the communication demand by reducing the "less significant" connection and communication nodes. This is the central idea behind the various modeling (including Large Language Models). As their training and usage times highlight, pruning is only possible when the weights are already known and (with some limitations) some nodes can be omitted. All nodes must be used during training, resulting in unacceptable high access times. However, pruning also removes valuable information, degrading the accuracy by enhancing running time. We cite again: "the running time decreases, but so does performance" [105]. (Our brain is sparsely connected, but its sparsity is defined genetically instead of the frequency of using the nodes.)

## 4. Biological Computing

When discussing the operation of biological computing, especially for a reader with a background in technical computing, one must always keep an eye on the differences in "implementations". There are several orders of magnitude differences in the operating speeds, and some other approximations commonly assumed in electronic technology are not valid in biological computing. The enormous signal propagation speed in metals makes the 'instant interaction' a good approximation for discussing electronic operation (although it worsens as the operating frequency increases). However, it is 'unsound' (as von Neumann said) when the signal propagation is a million times slower, in the range of $m/s$. Below, we use the notion 'current' in a sense slightly different from the one commonly used in the theory of electricity: the charge carriers are about 50,000 times heavier than the electrons, so their electric interaction must not be considered instant.

Furthermore, one must keep in mind that the charge transfer mechanism in electrolytes (and especially for the structures found in biology) is different from the one the audience is used to, so as E. Schrödinger has formulated [47], 'non-ordinary' laws [48] of science also describe the computational behavior of biology. At this level of abstraction, one does not need to consider biological details since "Despite the extraordinary diversity and complexity of neuronal morphology and synaptic connectivity, *the nervous systems adopts a number of basic principles*" [106]. Although we discuss the operation of neurons here, we must not forget that "what makes the brain a remarkable information processing organ is not the complexity of its neurons but the fact that it has many elements interconnected in a variety of complex ways” [12].

Biological computing is data- and event-controlled. The neuron’s elementary computing unit has an abstract representation as a self-controling *serial RC* oscillator [45]. Although it has many (several thousand) inputs, it uses only about a dozen of them in a single computing operation, appropriately and dynamically weighting and gating its inputs [24,107]. It has many (several thousand) inputs. A computing operation uses only about a dozen at a time, appropriately and dynamically weighting and gating its inputs [24,107].

Furthermore, the length of the axons between the neurons and the conduction velocity of the neural signals entirely define the time of the data transfer (in the msec range); all connections are direct. The transferred signal starts the subsequent computation as well (asynchronous mode). ”A preconfigured, strongly connected minority of fast-firing neurons form the backbone of brain connectivity and serve as an ever-ready, fast-acting system. However, full performance of the brain also depends on the activity of very large numbers of weakly connected and slow-firing majority of neurons.” [24]

The classic computing paradigm does not apply at all ("unsound" [50]) to biology because of the low conduction velocity of neural signals while they are transmitting and processing. The transfer time is much longer than the neuronal processing time. Within the latter, the internal delivery time (“forming a spike”) is much longer than the computing (collecting charge from the presynaptic neurons) itself. Omitting the computing time aside from transmission time would be a better approximation than the opposite one included in the theory. *The arguments and the result are temporal; the neural processing is analog and event-driven* [108].

Synaptic charges arrive when the time window is open, or artificial charges increase or decrease the membrane’s potential. A neuron has memory-like states [109], see also Figure 11 (implemented by different biophysical mechanisms), the computation can be restarted, and its result also depends on the actual neural environment and neuronal sensitivity. When all factors are known, the operation can be described. However, because of the enormous number of factors, their time dependence, the finite size of the neuron’s components, and the finite speed of ions (furthermore, their interaction while traveling on the dendrites), it is much less deterministic (however, not random!) than the technical computation. We learned that the neuron’s behavior can hardly be described mathematically and be modeled electronically. The inputs considered (those arriving within the time window) in the computation, their weights, and arrival times change between the operations, making it hard to calculate the result on the one hand. On the other hand (accompanied by the learning mechanism), it enables the implementation of higher-order neural functionality, such as intuition, association, redundancy, and rehabilitation.

### *4.1. State Machine*

Figure 10 illustrates our abstract view of a neuron, in this case, as a "state machine". Notice that the double circles are *stages* (*states* with event-defined periods) connected by bended arrows representing *instant stage transitions*. At the same time, at some other abstraction level we consider them *processes* that have a temporal course with their *event* handling. Fundamentally, the system circulates along the blue pathway and maintains its state (described by a single-stage variable, the membrane potential) using the black loops. However, sometimes it takes the less common red pathways. It receives its inputs cooperatively (controls the accepted amount of its external inputs from the upstream neurons by gating them by regulating its stage variable). Furthermore, it actively communicates *the time of its state change* (*not its state* as assumed in the so-called neural information theory) toward the downstream neurons in a process parallel with its mentioned activity.

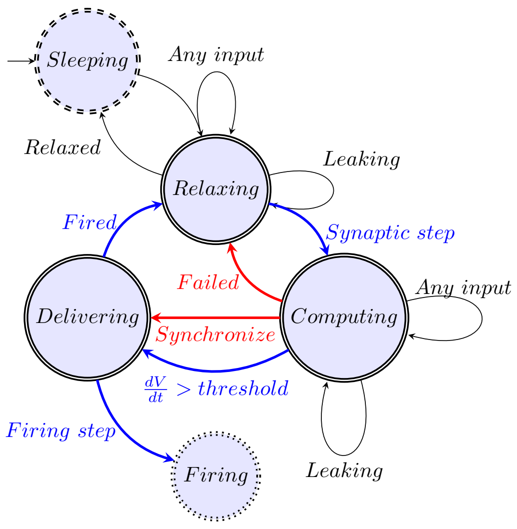

**Figure 10.** The model of neuron as an abstract state machine

Initially, a neuron is in the "Relaxing" stage, which is the ground state of its operation. (We also introduce a "Sleeping" or "Standby" helper stage, which can be imagined as a low-power mode [110] in the electronic or state maintenance mode of biological computing or "creating the neuron" in biology; a "no payload activity" stage.) The stage transition from "Sleeping" also *resets the internal stage variable* (the membrane potential) to the value of the resting potential. In biology, a "leaking" background activity takes place: it changes (among others) the stage variable toward the system's "no activity" value.

While computing, a current flows out from the neuronal condenser, so *the arrival time of its neural input charge packets (spikes) matters*. All charges arriving when the time window is open increase or decrease the membrane's potential. The neuron has memory-like states [109] (implemented by different biophysical mechanisms); the computation can be restarted, and its result also depends on the actual neural environment and neuronal sensitivity. Although the operation of neurons can be described when all factors are known, because of the enormous number of factors and their time dependence (including 'random' spikes), it is much less deterministic (however, not random!) than the technical computation. The inputs considered in the computation (those arriving within the time window), their weights, and arrival times change dynamically between the operations. On the one hand, this change makes it hard to calculate the result; on the other (accompanied by the learning mechanism), it enables implementing higher-order neural functionality, such as redundancy, rehabilitation, intuition, association, etc. Constructing solid electrolytes enables the creation of artificial synapses [111], and many biological facilities in reach, with the perspective of having a thousand times faster neurons, provide facilities for getting closer to the biological operation.

An event (in the form of a pulse of slow ions) arriving from the environment acts as a "look at me" signal, and the system passes to the "Computing" stage: an excitation begins. The external signal triggers a stage change and simultaneously contributes to the value of the internal stage variable (membrane voltage). During regular operation, when the stage variable reaches the critical value (the threshold potential), the system generates an event that passes to the "Delivering" stage and "flushes" the collected charge. In that stage, it sends a signal toward the environment (to the other neurons connected to its axon). After a fixed period, it passes to the "Relaxing" stage without resetting the stage variable. From this event on, the "leaking" and the input pulses from the upstream neurons contribute to its stage variable.

The process "Delivering" operates an independent subsystem ("Firing"): it happens simultaneously with the process "Relaxing", which, after some time, usually passes to the next "Computing" stage. Notice that *the stages "Computing" and "Delivering" mutually block each other, and the I/O operations happen in parallel with them*. They have temporal lengths, and they must follow in the well-defined order

"Computing"⇒"Delivering"⇒"Relaxing" (a "proper sequencing" [50]). Stage "Delivering" has a fixed period, stage "Computing" has a variable period (depends mainly on the upstream neurons), and the total length of the computing period equals their sum. The (physiologically defined) length of the "Delivering" period limits the neuron's firing rate; the length of "Computing" is usually much shorter.

In any stage, a "leaking current" changing the stage variable is present; *the continuous change (the presence of a voltage gradient) is fundamental for a biological computing system*. This current is proportional to the stage variable (membrane current); it is *not* identical to the fixed-size "leaking current" assumed in the Hodgkin-Huxley model [112]. The event that is issued when stage "Computing" ends and "Delivering" begins separates two physically different operating modes: inputting payload signals for computing and inputting "servo" ions for transmitting (signal transmission to fellow neurons begins and happens in parallel with further computation(s)).

There are two more possible stage transitions from the stage "Computing". First, the stage variable (due to "leaking") may approach its initial value (the resting potential), and the system passes to the "Relaxing" stage; in this case, we consider that the excitation "Failed". This happens when leaking is more intense than the input current pulses (the input firing rate is too low, or a random firing event starts the computing). Second, an external pulse [113] "Synchronize" may have the effect of forcing the system (independently from the value of the stage variable) to pass instantly to the "Delivering" stage and, after that, to the "Relaxing" stage. (When several neurons share that input signal, they will go to the "Relaxing" simultaneously: they get synchronized, a simple way of synchronizing low-precision asynchronous oscillators.)

Anyhow, a neuron operates in cooperation with its environment (the fellow neurons, with outputs distributed in space and time). It receives multiple inputs at different times (at different offset times from the different upstream neurons) and in different stages. In the "Computing" stage, the synaptic inputs are open, while in the "Delivering" stage, the synaptic inputs are closed (the input is ineffective). It produces multiple outputs (in the sense that the signal may branch along its path) in the form of a signal with a temporal course.

## *4.2. Conceptual Operation*

Concerning Figure 11, we subdivide neuron's operation into three stages (green, red, and blue sections of the broken diagram line), in line with the state machine in Figure 10. We start in the 'Relaxing' stage (it is a steady state, with the membrane's voltage at its resting value). Everything is balanced, and synaptic inputs are enabled. No currents flow (neither input nor output); since all components have the same potential, an output current has no driving force (there is no "leaking current" [112]).

The neuron has a stage variable (the membrane potential) and a regulatory threshold value. A threshold for *voltage gradient* exists instead of the *membrane's voltage* itself (the voltage gradient provides a 'driving force'). The voltage sensing is based on voltage gradient sensing (a simple electrometer), which phenomenon correlates with the value of the membrane's voltage. Given that physiological measurements (such as clamping) suppress the gradient, and, in a 'freezed' state, only the voltage is measured; this difference has remained hidden. Crossing the membrane's voltage threshold value upward and downward causes a stage transition from "Computing" to "Delivering" and from "Delivering" to "Relaxing", respectively. Another role of that regulatory value is to open/close the input synapses. Furthermore, when the value exceeds the threshold, an intense current starts to charge up the condenser, that later discharges.

Given that the neuron's operation resembles an *RC* oscillator, the capacitive current of the condenser changes its direction, changing the potential relative to the charge-up potential value to a value of the opposite sign. The time constant of the *RC* oscillator is set so that the rushed-in current generates a nearly critically damped oscillation (with a damping parameter about $\zeta = 0.35$).

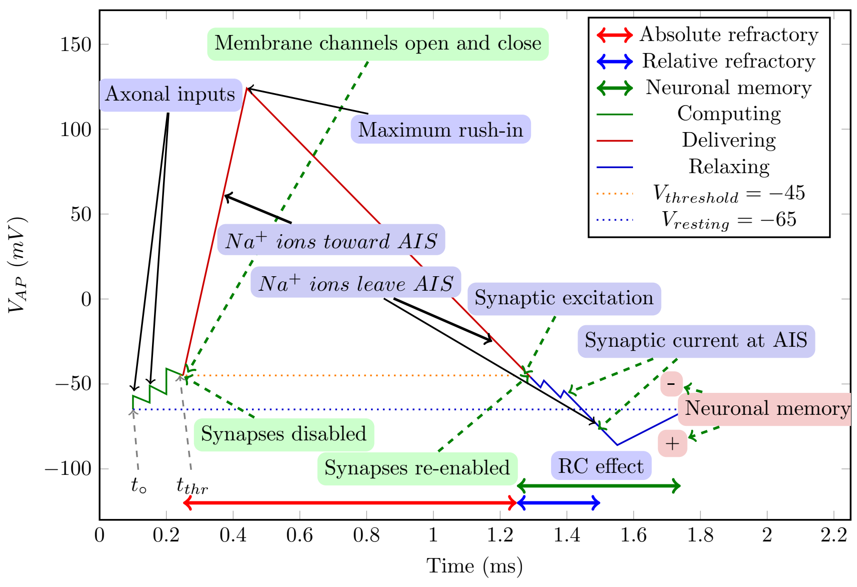

**Figure 11.** The conceptual graph of the action potential

Notice that all these processes happen with well-defined speeds, i.e., the different stages have well-defined temporal lengths. The length of the period "Delivering" is fixed (defined by physiological parameters), the length of "Computing" depends on the activity of the upstream neurons (furthermore, on the gating due to the membrane's voltage). Due to the finite speed, we discuss all operations in neuron's own "local time".

When the membrane's voltage decreases below the threshold value, the axonal inputs are re-opened, that may mean an instant passing to stage "Computing" again. The current stops only when the charge on the membrane disappears (the driving force terminates), so the current may change continuously, changing the voltage on the circuit's output. The time of the end of the operation is ill-defined, and so is the value of the membrane's voltage when the next axonal input arrives. *The residual potential acts as a (time-dependent) memory*, with about a *msec* lifetime; see Figure 11.

### 4.2.1. Stage 'Computing'

The neuron receives its inputs as 'Axonal inputs'. For the first input in the 'Relaxing' stage, the neuron enters the "Computing" stage. The time of this event is the base time used for calculating the neuron's "local time". Notice that to produce the result, the neuron cooperates with upstream neurons (the neuron gates its input currents). One of the upstream neurons opens computing, and the receiving neuron terminates it.

The figure aso discovers one of the secrets of the marvelous efficiency of neuronal computing. *The dynamic weighting of the synaptic inputs plus adding the local memory content happens analog, per synapse, and the summing happens at the jointly used membrane. The synaptic weights are not stored for an extended period.* It is more than a brave attempt to accompany (in the sense of technical computing) a storage capacity to this time-dependent weight and a computing capacity to the neuron's time measurement.

### 4.2.2. Stage 'Delivering'

In this stage, the result is ready: the time between the arrival of the first synaptic input and reaching the membrane's threshold voltage is measured. No more input is desirable, so the neuron closes its input synapses. Simultaneously, the neuron starts its "servo" mechanism: it opens its ion channels and an intense ion current starts to charge the membrane. It is an 'instant' current. The voltage on the membrane quickly rises, and it takes a short time until its peak voltage is reached. Given that the charge-up current is instant and the increased membrane voltage drives an outward current, the membrane voltage gradually decays. When the voltage drops below the threshold voltage, the neuron re-opens its synaptic inputs and passes to stage “Relaxing”: it is ready for the next operation. The signal transmission to downstream neurons happens in parallel with the recent "Delivering" stage and the subsequent "Relaxing" (and maybe "Computing") stages.

#### 4.2.3. Stage ’Relaxing’

In this stage, the neuron re-opens its synaptic gates. Recall that the ion channels generating an intense membrane current are already closed. The neuron passes to stage ’Relaxing’ and is ready for a new computation: the previous result is under delivering (parallelly, independently), and the axonal inputs are open again. However, the membrane’s potential may differ from the resting potential. A new computation begins (the neuron passes to the stage ’Computing’) when a new axonal input arrives. Given that the computation is analog, a current flows through the AIS, and the result is the length of the period to reach the threshold value. The membrane voltage plays the role of an accumulator (with a time-dependent content): a non-zero initial value acts as a short-time memory in the subsequent computing cycle. The local time is reset when a new computing cycle begins, but not when eventually the resting potential reached. Notice that the same stage control variable plays many roles: the input pulse writes a value into the memory (the synaptic inputs generate voltage increment contributions which decay with time, so the different decay times set a per-channel memory value while simultaneously the weighted sum is calculated).

#### 4.2.4. Synaptic Control

As discussed, controling the operation of its synapses is a fundamental aspect of neuronal operation. It is a gating and implements an ’autonomous cooperation’ with the upstream neurons. The neuron’s gating uses a ’downhill method’: while the membrane’s potential is above the axonal arbor’s, the charges cannot enter the membrane. When the membrane’s voltage exceeds the threshold voltage, the synaptic inputs stop and restart only when the voltage drops below of that threshold again. The synaptic gating makes interpreting neural information and entropy, as discussed it in [7], at least hard.

#### 4.2.5. Operating Diagrams

Figure 12 (produced by the simulator) shows how the described physical processes control neuron’s operation. In the middle inset, when the membrane’s surface potential increases above its threshold potential due to three step-like excitations opening the ion channels, $Na^+$ ions rush in instantly and create an exponentially decreasing, step-like voltage derivative that charges up the membrane. The step-like imitated synaptic inputs are resemblant to the real ones: the incoming PSPs produce smaller, rush-in-resemblant, voltage gradient contributions. The charge creates a thin surface layer current that can flow out through the AIS. This outward current is negative and proportional to the membrane potential above its resting potential. In the beginning, the rushed-in current (and correspondingly, its potential gradient contribution) is much higher than the current flowing out through the AIS, so for a while, the membrane’s potential (and so: the AIS current) grows. When they get equal, the AP reaches its top potential value. Later, the rush-in current gets exhausted and its potential-generating power drops below that of the AIS current; the resulting potential gradient changes its sign and the membrane potential starts to decrease.

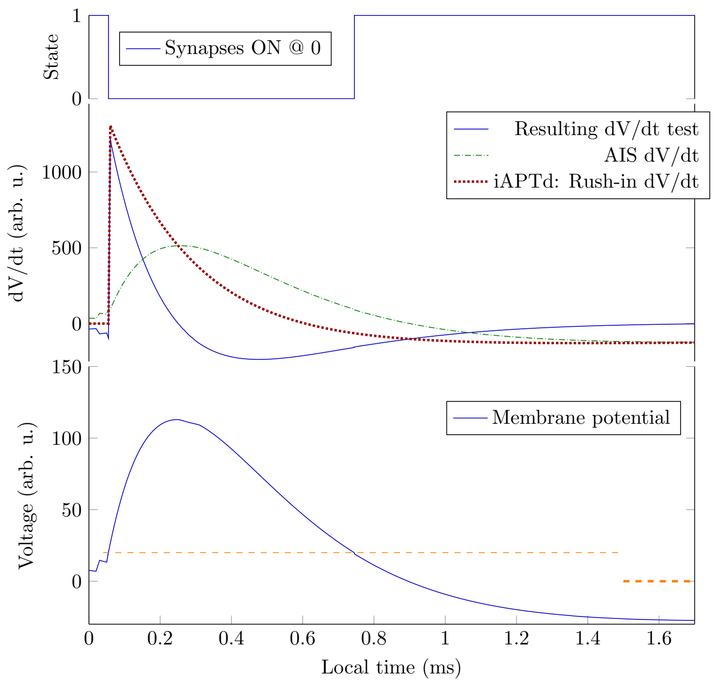

**Figure 12.** How the physical processes describe membrane's operation

In the previous period, the rush-in charge was stored on the membrane. Now, when the potential gradient reverses, the driving force starts to decrease the charge in the layer on the membrane, which, per definitionem, means *a reversed current; without foreign ionic current* through the AIS. This is a *fundamental difference between the static picture* that Hodgkin and Huxley hypothesized [112] *and biology uses, and the dynamic one that describes its behavior*. The microscopes' resolution enabled to notice AIS only two decades after their time; their structure could be studied only three more decades later [114]. In the past two decades, it has not been built into the theory of neuronal circuits. The correct equivalent electric circuit of a neuron is a serial, instead of a parallel, oscillator (see Section 4.3.3), and its output voltage is defined dynamically by its voltage gradients instead of static currents (as physiology erroneously assumes). In the static picture, the oscillator is only an epizodist, while in the time-aware (dynamic) picture, it is a star.

Notice also that *only the resulting* $\frac{dV}{dt}$ *APTD disappears* with time. Its two terms are connected through the membrane potential. As long as the membrane's potential is above the resting value, a current of variable size and sign will flow, and the output and input currents are not necessarily equal: the capacitive current changes the game's rules.

The top inset shows how the membrane potential controls the synaptic inputs. Given the ions from the neuronal arbor [115,116] can pass to the membrane using the 'downhill' method, they cannot do so if the membrane's potential is above the threshold. The upper diagram line shows how this gating changes in time's function.

### 4.2.6. Classic Stages

We can map our 'modern' (dynamic) stages to those 'classic' (static) stages, and we can see why defining the length of the Action Potential is problematic. The effect of slow current affects the apparent boundary between our "Delivering" stage and "Relaxing" stages. Classical physiology sees the difference and distinguishes 'absolute' and 'relative' refractory periods with a smeared boundary between them. Furthermore, it defines the length of the spike with some characteristic point, such as reaching the resting value for the first or second time or reaching the maximum polarization/hyperpolarization. Our derivation of the stages (see Figure 11) defines clear-cut breakpoints between them.

We can define the spike length as the sum of the variable-size length of periods "Computing" and fixed-size period "Delivering". The "absolute refractory" period is defined as the period when the neuron membrane's voltage keeps the gates of the synaptic inputs closed (the value of membrane voltage is above their threshold). That period is apparently extended (and interpreted as a "relative refractory" period) by the period when, although the gating is re-enabled, but the slow current did not yet arrive at the Axon Initial Segment where it contributes to the measured AP, see Figure 11. *Only one refractory period exists, plus the effect of the slow current.*

One must be careful when extrapolating results derived for a single "neuronal link" to a set of neurons. If we consider that the number of input spikes carries the information, it is typical that several spikes arrive at a neuron, and only one spike is produced: "A single neuron may receive inputs from up to 15,000–20,000 neurons and may transmit a signal to 40,000–60,000 other neurons" [82]. "In the terminology of communication theory and information theory, [a neuron] is a multiaccess, partially degraded broadcast channel that performs computations on data received at thousands of input terminals and transmits information to thousands of output terminals by means of a time-continuous version of pulse position. Moreover, [a neuron] engages in an extreme form of network coding; *it does not store or forward the information it receives but rather fastidiously computes a certain functional of the union of all its input spike trains* which it then conveys to a multiplicity of select recipients" [46]. In this way, some "information loss" surely takes place. If we consider that the ISI carries information and a neuron takes into consideration in its "computing" only the spikes which arrive within an appropriate time window, some information is lost again. Even *the result of the computation, a single output spike, may be issued before either of the input spikes was entirely delivered* – one more reason why the notion of information should be revisited.

## 4.3. Electrical Description

### 4.3.1. Hodgkin-Huxley Model

More than seven decades ago, Hodgkin and Huxley [112] elaborated a genial hypothesis that the operation of neural cells can be described in terms of electricity: currents, voltages, and an oscillator. Although it was an enormous step toward describing the brain's operation – mainly due to the lack of knowledge that became available decades later- their first step needs corrections. As detailed in [117] we need to fix some errors. Although their view that electric processes are in the background, was correct, we must correct that the charge carriers in the circuit are slow ions instead of fast electrons, the Coulomb-repulsion between the carriers matters (the current can be modeled as viscous charged fluid [118]), 'non-ordinary laws' [47] describe the processes [48] in the cell and the basic neuronal circuit is a serial oscillator instead of a parallel one. Unfortunately, discussions, models, and equations based on their original hypotheses are wrong. However, with the major changes in Section 4.3.3, we can perfectly describe, without ad-hoc assumptions, the electrical behavior of neurons, including their computing abilities.

### 4.3.2. Electrotonic Model

Even their unfortunate illustrative idea of 'equivalent circuits' led to analyzing "electrotonic (electronic circuit equivalent) modeling of realistic neurons and the interaction of dendritic morphology and voltage-dependent membrane properties on the processing of neuronal synaptic input" [119]; that is, to study a simulated neuron built from discrete electronic components. However, the idea needs to put together many component neurons "raises the possibility that the neuron is itself a network": the static picture cannot describe a dynamical behavior. On the one side, such an idea misguides the neurophysical research (since a fake neural system is scrutinized, the validity of the approach is questionalized [5]). Conversely, since electroengineers understand the neuronal operation from the wrong model, it also misguides building neuromorphic architectures.

### 4.3.3. Physical Model

The so called equivalent circuit of the neuron is a differentiator-type $RC$ oscillator circuit (it is erroneously stated in biophysics that it is an integrator-type oscillator). The synaptic currents can increase the membrane's voltage above its threshold, so an intense current starts that suddenly increases the membrane's voltage to a peak voltage, and then the condenser discharges. The circuit produces a damped oscillation, and after a longer time, the membrane voltage returns to its resting value. While the membrane's voltage is above its threshold value, the synaptic inputs are inactive (the analog inputs are gated).

In our model, the neuron is represented as a constant capacity condenser $C_M$ and a constant resistance $R_M$. They are connected to form a serial (differentiator-type) oscillator with time constant $R_M C_M$. However, there are significant differences between the discrete elements used in the theory of electricity and our neuron model. The charge transfer mechanism is entirely different; moreover, the ions must move physically (as opposed to the "electron cloud" model). The charge carriers are slow (typically positive) ions, which are about 50,000 times heavier than electrons, while the electrostatic force between the carriers is the same as in the case of electrons. Due to the slow ions, the "instant interaction" model used in the theory of electricity cannot be used. However, using current generators of special form, one can imitate the phenomena of classic electricity. The synaptic currents arrive through an axonal arbor where they slow down according to the Stokes-Einstein relation (the cross-section of the arbor is several times higher than that of the axon). The arbor acts as a buffer condenser with capacity $C_{A,i}$, representing a voltage at the junction point and enabling gating of the synaptic inputs. Since the ions can move only using the "downhill" method, synaptic current can flow only if the voltage on the arbor's exit is above the membrane's voltage. Given the low speed of the ions, this condition must be interpreted dynamically.

Concerning Figure 11, given that slow ion currents deliver the charge on the membrane's surface, the charge propagation takes time and so after the charge-up process, the synaptic inputs reach the output of the circuit about a tenth of a millisecond later. Their arrival provides input for the next computing cycle. Given that the calculation is implemented as integrating charge on the condenser, a potential deviating from the resting one implements a neuron-level memory.

Although the implementation of a neuronal circuit works with ionic currents and has a charge transmission mechanism drastically different from the one based on free electrons used in metals, the operation can align (to some measure) with the instant currents of physical electronics. (As we suggest, the effect of the slow current and finite size can be reasonably imitated with current generators generating a special current shape.) The input currents in a biological circuit are a large rush-in current through the membrane in an extremely short time and much smaller gated synaptic currents arriving at different times through the synapses. Those currents generate voltage gradients, and those gradients direct the operation of the $RC$ oscillator. As it is well known from the theory of electric circuits, the output voltage measured on the output resistor is

$$V_{out}^{Differentiator} = RC\frac{dV_{in}}{dt} \tag{1}$$

where the input voltage is

$$\frac{d}{dt}V_{in} = \sum \frac{d}{dt}V_{IN}^{Component} - \frac{d}{dt}V_{OUT}^{AIS} \tag{2}$$

that is the (temporally gated) sum of the input gradient that the currents generate plus the gradient of the output current through the AIS [114,120]. The latter term can be described as

$$\frac{d}{dt}V_{OUT}^{AIS} = \frac{V_{internal} - V_{external}}{R} \tag{3}$$

The input currents (although due to slightly differing physical reasons for the different current types) are described by an analytic form

$$I_{in} = I_o * (1 - \exp(-\frac{1}{\alpha} * t)) * exp(-\frac{1}{\beta} * t) \tag{4}$$

where $I_o$ is a current amplitude per input, $\alpha$ and $\beta$ are timing constants for current in- and outflow, respectively. They are (due to geometrical reasons) approximately similar for the synaptic inputs and differ from the constants valid for the rush-in current. To implement such an analog circuit with a conventional electronic circuit, voltage-gated current injectors (with time constants of around 1 *ms*) are needed. The corresponding voltage derivative is

$$\frac{dV_{in}}{dt} = \frac{I_o}{C} * \left( \frac{1}{\alpha} * exp(-\frac{1}{\alpha} * t - \frac{1}{\beta} * t) - \frac{1}{\beta} * exp(-\frac{1}{\beta} * t) * exp(1 - exp(-\frac{1}{\alpha} * t)) \right) \tag{5}$$

In the simplest case, the resulting voltage derivative comprises only the rush-in contribution described by Equation (5) and the AIS contribution described by Equation (3). The gating implements a dynamically changing temporal window and dynamically changing synaptic weights. The appearance of the first arriving synaptic gradient starts the “Computing” stage. The rush-in gradient starts the “Delivering” stage. The charge collected in that temporal window is

$$Q(t) = \int_{t_{0,i}}^{t_{thr}} dt \underbrace{\sum_{i}}_{t_{0,i} \le t \le t_{thr}} I_{syn,i}(t) \tag{6}$$

The total charge is integrated in a mathematically entirely unusual way. The neuron (though the integration time window) selects which ones out of its upstream neurons can contribute to the result; the individual contributions are a bargain between the sender and the receiver: the neuron decides when a contributing current terminates, but the upstream neuron decides when it begins. *The beginning of the time window is set by the arrival of a spike from one of the upstream neurons, and it is closed by the neuron when the charge integration results in a voltage exceeding the threshold.* The computation (a weighted summation) is performed on the fly, exluding the obsolete operands. The only operation that a neuron can perform is that unique integration. Its result is the reciprocal of the weighted sum, and it is represented by the time when a spike is sent out (*when* that charge on the membrane’s fixed capacity leads to reaching the threshold). Notice that the time is defined on the local time scale and is interpreted similarly by the downstream neurons.

Figure 12 (resulted by our simulator) shows how the described physical processes control neuron’s operation. In the middle inset, when the membrane’s surface potential increases above its threshold potential due to three step-like excitations opens the ion channels, ions rush-in instantly and create an exponentially decreasing, step-like voltage derivative that charges up the membrane. The step-like imitated synaptic inputs resemble the real ones: the synaptic inputs produce smaller, rush-in-resemblant, voltage gradient contributions. When integrated as Equation (1) describes, the resulting voltage gradient produces the Action Potential shown in the lower inset. Crossing the threshold level controls the gating signal as shown in the top inset.

Notice how nature implemented "in-memory computing" by weighted parallel analog summing. Notice nature’s principles of reducing operands and noise: the less important inputs are omitted by timing. The rest of the signals are integrated over time. The price paid is a straightforward computing operation: the processor can perform only one single type of operation. Its arguments, result, and neuronal-level memory are all time-dependent. Furthermore, the operation excludes all but (typically) a dozen of neurons from the game. It is not necessarily at all that the same operands, having the same dynamic weights are included in consecutive operations. It establishes a chaotic-like operation [121] at the level of the operation of neuronal assemblies [23,122,123], but also enables learning, association, rehabilitation, and similar functionalities. Since the signal's shape is identical, integrating them in time means multiplying it by a weight (depending non-linearly on time), and the deviation from the steady-state potential represents a (time-dependent) memory. The significant operations, the dynamic weighting, and the dynamic summing are performed strictly in parallel.

Essentially, this is why, from the point of view of technical computation, "we are between about 7 and 9 orders of magnitude removed from the efficiency of mammalian cortex" [14]. The statement is about using only 1% of the available HW; using more HW increases power consumption but results in no increase in absolute performance. It was early recognized [83] that for a given workflow, when the nominal performance increases, the payload performance reaches a maximum, and then it decays. The parameters of the course of performance also depend on the available parallelization technology, but as confirmed for supercomputers and distributed computing [84,124], measured for artificial neural networks [85], and theoretically understood [86], for most *practical workflows*, a few dozens of technical computing units can be reasonably used (they contribute significant increase to the absolute performance). In biology, this number is in the order of a hundred billion. In the recent generative AI systems, due to "pruning", a variable number of computing units are excluded, and some more are effectively omitted due to the real-time reply constraint. Even for "racing" workflows, the efficiency decreases to 50% of the expected peak performance; see the actual [125] stories.

*Timing relations*

Von Neumann made a careful feasibility analysis [50] and warned: "*6.3 At this point, the following observation is necessary. In the human nervous system, the conduction times [transmission times] along the lines (axons) can be longer than the synaptic delays [the processing times], hence our above procedure of neglecting them aside from $\tau$ [the processing time] would be unsound. In the actually intended vacuum tube interpretation, however, this procedure is justified: $\tau$ is to be about a microsecond, an electromagnetic impulse travels in this time 300 meters, and as the lines are likely to be short compared to this, the conduction times may indeed be neglected. (It would take an ultra-high frequency device – $10^{-8}$ seconds or less – to vitiate this argument).*" Unfortunately(?), today's devices have orders of magnitude higher frequency.

Von Neumann was aware of the facts and the technical development: "We reemphasize: This situation is only temporary, only a transient standpoint ... After the conclusions of the preliminary discussion, the elements will have to be reconsidered" [126]. Unfortunately, the computing paradigm and the computer as a device were too successful, so *the elements have never been reconsidered*, except [21].

In section 5.4, von Neumann told "We propose to use the delays $\tau$ as absolute units of time which can be relied upon to synchronize the functions of various parts of the device. The advantages of such an arrangement are immediately plausible". When choosing such an absolute time unit, a "worst-case" timing must be selected, that *inherently introduces performance loss* for the "less wrong" cases. When looking at Figure 1, we see that *the total execution times of operation (instead of processing time only) must be as uniform as possible*. (It is worth having a look at Figure 11: the "Computing time" dwarfs aside the internal delivery time. A fundamental principle that biomorph technology did not discover. Biology operates with nearly critically damped circuits, while technical computing applies energy-wasting square-waves.) If their *processing time* (different complexity) or *transfer time* (different connection technology or *signal propagation length*) differs, the synchronization inherently introduces some performance loss. The processing elements will be idle until the next clock pulse arrives.

Developing and using asynchronous computing methods is not simple, especially when its design is made by experts with synchronous solutions in mind and is built from technological components manufactured for synchronous computing. Mainly, this contradiction prevented the development of genuinely parallel computing despite the vast need, support, and collective efforts [64].

## 5. Biological Learning Vs Machine Learning

Learning is a critical factor in life's evolution and individual survival. "*The broad definition of learning: use present information to adjust a circuit, to improve future performance*" [127] requires a deeper understanding of what this "present information" (and information at all) is; furthermore, it implies that *learning is a temporal process*. Information handling, a highly intricate process, involves a series of operations such as coding, decoding, transferring, storing, and retrieving. Each of these processes is essential for effective information management.

During learning, the organ uses the past information stored in its internal state variables. However, "*we should not seek a special organ for 'information storage' — it is stored, as it should be, in every circuit*" [127] (our discussion above explains how it is implemented). The definition also suggests considering learning (and, because of this, information processing) as a temporal process: the organs have a sensing time, a processing time, and a storage access time (despite that "*information stored directly at a synapse can be retrieved directly*" [127]) and a transfer time (conduction time) to the next computing organ. Based on the time-aware computing paradigm, the general model of computing [21] correctly describes the general learning process (underpinned by anatomical evidence).

## 5.1. Biological Learning

In biology, the interaction speed (conduction velocity) is a million times lower than the apparent propagation speed of free electrons in wires, so the 'spatiotemporal' behavior of the neuronal operation is commonly accepted (although it is handled by using an inappropriate method, using functions with separable time variable). We published our ideas about abstracted biological learning in [45,107], so we only briefly summarize our theses.

A spike arriving at a synapse opens the voltage-gated channels, enabling charge delivery to the membrane [37,106]. That is, the operand itself generates the 'Operand available' signal, and the arrival of the first operand generates the signal 'Begin computing'. The neuron's membrane makes the computation: it integrates the received charge until it reaches its threshold potential and generates the 'End computing' signal. When a signal 'End computing' is received inside the neuron membrane, a 'Refractory' period begins (prepares for a new computing operation), and a few microseconds at the AIS, a spike is prepared ('Signal delivery'). The 'Signal transfer' period starts when the spike starts towards its destination neuron, outside the source neuron. All signals carry synchronization signals; biology uses no central clock for synchronization [128].

Biology can modulate the interaction speed; delivering the same charge with a higher conduction speed means a higher synaptic current, an aspect that remains out of sight in technical implementations. The information is the length of the beginning time of the 'Computing cycle' and the end time of 'Signal delivery'. That is, biology uses *asynchronous auto-synchronization* [129] and uses a *mixed-mode digital/analog computing method*.

A neuron has two ways to adjust its spiking time: it uses more charge from its synaptic input or a higher current. In the first case, the neuron increases the width of its processing time window by increasing the local concentration of neurotransmitters; this mechanism is commonly mimicked in machine learning by modifying the "synaptic weight" $W_i$ of the $i$-th synapse. In the second case, it increases the synaptic current (increases the delivery speed), so the same amount of charge arrives at an earlier time, a mechanism that has no equivalent in machine learning. The dynamic balance of the two mechanisms enables learning and forgetting, replacing, redundancy, rehabilitation, etc. *Noticeably, in both cases, only the receiver neuron adjusts itself; the sender neurons need not adapt themselves (there is no direct feedback).* "The first property, by virtue of which the nerve cells react to the incoming impulse . . . we call *excitability*, and . . . changes arising . . . because of this property we shall call changes due to excitability. The second property, by virtue of which certain permanent functional transformations arise in particular systems of neurons as a result of appropriate stimuli or their combination, we shall call *plasticity* and the corresponding changes plastic changes." [12], page 37. That is, the changes seen can theoretically explained by the handling of operating time.

The effect in both cases is the same: the neuron reaches its threshold earlier. The first method is quick (in the millisecond range) but energy-wasting: keeping the concentration gradient requires much ion pumping. The second method is slow (in the range of days to weeks) and needs anatomical changes, but the operation needs lower energy consumption. That is, biology finds the best timing quickly (short-term learning), but if the external condition persists, it finalizes storing that information:

myelinates the corresponding axon (long-term learning; for anatomical evidence, see cited references in [45]). The evidence that neural networks must "invest" in information processing is known; see, for example, the chapter "Pricing Neural Information" in [130].

These two mechanisms are the two sides of the same coin. Biological neurons adjust their processing and transfer times. Biology stores information locally by adjusting time (a dynamic form). *Any study attempting to grasp biology's information storage and processing in some static form fails.* Notice that some learning details are pre-programmed, and the learned new knowledge is equivalent to the pre-programmed (maybe genetic) one. Furthermore, learning is a native and lifelong ability, as the elegant experiment [131] proved.

Biology uses cooperating neurons (so-called assemblies [23]). Consequently, the fundamental assumption of a Poisson model that spikes are generated independently is not fulfilled. Furthermore, the presence of a spike is a digital datum (one bit); the rest is analog. Given that the estimated information content of a spike is between one and three bits [127], the analog information may even dominate. *However, the mathematical theory's claim that timing delivers information is missing. Using information theory for neural information transfer needs revision; see [7] (for a definition of neural information, see [130]).*

## 5.2. Machine Learning

The most attractive feature of biological learning is that the ability to learn seems to be a native feature of the network of circuits, which can cope with the enormous amount of data from their environment without needing programming efforts. Computing technology attempts to organize various numbers of computing units (ranging from a few-neuron "toy" systems to supercomputers to many-billion parameter AI systems) into networks and attempts to mimic neuronal operations, including learning. To operate those networks, HW/SW solutions, and mathematical methods have been developed (more or less) mimicking biology.

By analogy, and to distinguish it from biological learning, the methods used in those networks are called 'machine learning' (sometimes mistakenly identified as "artificial intelligence" [132]). Those networks are built from components designed for processor-based computing, and those "biology-mimicking" systems are missing essential features of biological computing. Under these conditions, those systems can show remarkable success in some very specialized fields (such as playing a game or identifying a pattern). However, in general, given that the performance scaling is strongly non-linear [86], early noticed that "*core progress in AI has stalled in some fields*"[133].

## 5.3. Comparing Learnings and Intelligences

Although it is challenging to list the crucial differences between technological components attempting to mimic a biological component, furthermore to separate the HW and SW issues, some of the most important ones are mentioned below. It is at least complicated, if possible, to define the meaning of the word "intelligence" and dozens of meanings are used around [134,135]. The fundamental questions to reply are: "Whether it is implemented by molecules, cells, liquid crystals, silicon, or digital code, the essential operations of understanding are the same. Can the system acquire information external to itself? Can it generate an internal model of the external world by encoding information about it such that it can make predictions and inferences?".

The confusion starts at a much lower level. "At one extreme, the 'cognitive' in cognitive neuroscience has replaced the older term *information processing*. At the other extreme the term 'cognition' refers to those higher level processes fundamental to the formation of conscious experience. In common parlance, the term 'cognition' means thinking and reasoning." [12]. At the lowest level, both implementations do information processing. However, even at their lowest level, they process differently interpreted information on different structures using different methods. The notion of 'cognition' (similarly to 'conscience', and other notions) is not transferable between those implementations.

The principal operating difference is the sequential operation of technical systems that directly affects imitating biological operations. One of the primary motivations for using neural networks was the demand for processing actions on the correct biological time scale: “Many theoretical neurobiologists have turned to different types of models that include parallel processing, which they call neural networks.” [12], page 37. "The branch of computer science known as artificial intelligence originally used serial processing to simulate the brain’s cognitive processes—pattern recognition, learning, memory, and motor performance. These serial models performed many tasks rather well, including playing chess. However, they performed poorly with other computations that the brain does almost instantaneously" [12], page 37. More differences stemming from their different ’technology’ are discussed in [136].

Special care must be taken when handling signals ’Begin computing’, ’End computing’, and ’Signal delivery’. (Modern accelerating solutions in implementations, such as pipelining, ’out of order’ execution, branch predicting, etc., make description details much more complex, but – since they affect the result of the operation – the logical need to consider these signals persists.) An operating unit always has input signal levels for its operands. It provides output signal levels for its results, but the values of those signals correspond to their expected values only if the timing signals are appropriately aligned.

Chained computing operations must be considered for real-life computations, and their phases must be appropriately synchronized. Synchronization can be accomplished by different means. The operation’s availability signal can be produced using a central control unit (synchronous operation) or on a per-operand basis (asynchronous operation). Most of today’s technical computing systems use a central clock. As a sign of admitting that *the commonly used synchronized operating mode is disadvantageous in non-dispersionless systems*, the idea of asynchronous operation is around [11,34,44,71]. Central synchronization means that signals ’Operand available’, ’Begin computing’, ’Result available’, etc., are derived from a central clock signal. Signal ’Refractory’ is implemented as a short internal reset of circuits immediately before the falling edge of the synchronous signal. Similarly, ’Signal delivery’ must finish before the falling edge of the synchronous signal arrives. Given that the’Signal transfer’ length is undefined, in technical designs, transfer time is not included; that is, immediate interaction (infinitely large transfer speed) is assumed. At the inter-component level, the "skew" of the clock signal leads to the introducing of clock domains and clock distribution and causes dozens of percentage losses in energy efficiency [22,137].

Both effects increase the dispersion of clock signals and are majorly responsible for the experienced inefficiency [56] of computing. We can add that the appropriate synchronization needs special care when using accelerated and parallelized computations. Similarly, one must consider the temporal alignment when considering results from feedback and recurrent relations.

In biology, "*we should not seek a special organ for ’information storage’ — it is stored, as it should be, in every circuit*". The information "*is stored directly at a synapse and can be retrieved directly*" [127]. The idea of "*in-memory computing*" originates from here. It is forgotten, however, that storing and processing data cannot be separated (and in biology, is not). Furthermore that that principle enables only making simple computing.

In technical computing, we have technology blocks, among others, memory chips. Registers are essentially another memory with short addresses and access times; simulating data stored directly at synapses requires many registers. *Using conventional "far" memory instead of the direct register memory introduces a worse performance of two to three orders of magnitude. Summing synaptic inputs means scanning all registers, which requires several thousands of machine instructions and introduces another three to four orders of magnitude worse performance.* In addition, the correction term, calculated from the gradient, is distributed between all upstream neurons, whether they affected or not the result; it decreases again about two to three orders of magnitude.

From a computational point of view, biological axons represent a private connection between computing units, but billions of such (shallow speed) "buses" work in parallel, without contention. The usual technological implementation is using a single, high-speed bus, which must be "owned" for every communication. Using a serial bus means spending most of the processing time with arbitration.

Given that the needed arbitration limits the transfer time (increases with the communication intensity instead of the number of neurons!) rather than by the bus speed, using a single high-speed bus is at least questionable in large-scale systems: "*The idea of using the popular shared bus to implement the communication medium is no longer acceptable, mainly due to its high contention*." [95] Bus arbitration, addressing, and latency prolong the transfer time (and decrease the system's efficacy). This type of communicational burst may easily lead to a "communicational collapse" [15], but it may also produce unintentional "neuronal avalanches" [96]. As discussed in [45], the shared medium becomes the dominant contributor to computing's time consumption at many communicating units. Given that conventional processors are implemented in SPA, as Amdahl coined the wording [79], I/O instructions implement the communication between them. We can hope for better performance only in a drastically different approach [57,58].

The issues above led to decreasing communication as much as possible, including different methods for "pruning" [104], and the same idea is behind the various modeling (including Large Language Models). As their training and usage times highlight, pruning is only possible when the weights are already known and (with some limitations) some nodes can be omitted. All nodes must be used during training, resulting in unacceptable high access times. However, pruning also removes valuable information, degrading the accuracy by enhancing running time. We cite again: "the running time decreases, but so does performance" [105]. (*Our brain is sparsely connected, but its sparsity is defined genetically instead of the frequency of using the nodes*.)

As discussed in connection with biological neurons, signal timing is crucial in a network's operation. SW methods, not handling the simulated time explicitly, cannot align computing constraints properly. The usual SW organization is prepared to imitate single-thread processes. Some software libraries (such as SystemC [138,139]) which provide a unique engine enabling event scheduling on top of the OS's event scheduling. However, handling events is only possible through using services of the OS, which is very expensive in terms of non-payload instructions [41,42]. Handling events at a reasonable non-payload performance requires an entirely different architecture [58,117]. As admitted, "*building this new hardware [neuromorphic computing] necessitates reinventing electronics*"; furthermore, "*more physics and materials needed*" [34]. 'Rebooting computing' is needed from the ground up. Furthermore, as discussed above, rebooting the brain's dynamic operation needs to use the correct, 'non-ordinary' as E. Scrödinger coined, laws of science [48].

Furthermore, "it is also possible that non-biological hardware and computational paradigms may permit yet other varieties of machine intelligence we have not yet conceived" [63,140]. Consequently, we can learn that biological brains are more efficient learners than modern ML algorithms due to extra 'prior structure"' [140].

## 5.4. Imitating Neural Computations

Neural networks perform complex and ill-defined computations, including biological ones, by technical means. The technical neurons are connected and organized into so-called layers and forward data to each other, as shown in [141] and Fig 5. Neurons in the input layer receive the input data, from which (after weighting them, imitating their synaptic sensitivity) the input data of the next layer are calculated and forwarded to it. Of course, a layer can only perform its computations if the previous layer completed its computation. *Von Neumann required that the two computing windows of the layers must not overlap, and a data transfer time window must exist between them*. Systems using CPU (and other sequential systems) automatically fulfill this requirement. However, it is usual to perform the computations of the layer (to speed them up) using a Graphic Processing Unit (GPU) in parallel.

### 5.4.1. Using Accelerators

Using a sequential bus, by definition, excludes the simultaneous arrival of data and changes the arguments' delivery time. However, its genuinely disadvantageous consequence is that *the order of delivered messages is no longer defined*: one cannot apply Hebb's rule of learning anymore. In the case of more complex networks (comprising many neurons or many neural layers), one layer node may receive data from a previous layer node somewhat later than the others. Suppose the individual layers use the same high-speed bus (unlike as shown in Figs 5 and 7); the data traffic from different layers may block each other, even if separate GPUs perform the computations; GPU-foreign traffic also loads the bus.

In addition to wasting valuable time with contenting for the exclusive use of the single serial bus, utilization of the bit width (and package size) is also inefficient. Temporal precision "encodes the information in the spikes and is estimated to need up to three bits/spike" [130]. The time-unaware time stamping of artificial spikes does not encode temporal precision, given that the message comprises the *sending* time coordinate of the event instead of the relative time to a local synchrony signal (a base frequency); furthermore, the conduction time is not included in the message. The time coordinate of *receiving* the time-stamped event is missing. Time stamping misses the biological method of coding information [7].

The discussed operation (from a distance) resembles actual neural operation and is essentially data-controlled. Real biological neurons learn autonomously: they adjust their synaptic sensitivity based on the timing relations of the data arriving at their inputs. On the contrary, one must train artificial neurons, typically using the back-propagation method; see [141] and Figure 7. During that operation, the program (which works in instruction-driven mode) supervises the learning process, performs computations, and sends its results through the network and the bus in opposite directions. In the figure, the directions of data transmission are opposite in the case of 'forward' and 'backward' directions; the time is unidirectional. One can operate when the computation in the "forward" direction has already terminated (*the time windows overlap*, and they have no transfer time window between). Furthermore, the above restriction is valid for the computations inside the layers.

*Although the method has no biological relevance, it was said "deeply suspicious", and its inventor suggested using his breakthrough method should be dispensed with [142]*, a light-minded acceleration facility shined up for making the wrong method faster and even worse. Independent GPUs in the two layers can simultaneously perform computations in the "forward" and "backward" directions. However, changing the weights during the "backward" computation also partially affects the "forward" computation: the operation overwrites part of the weights unpredictably. *The backward computation works with uninitialized values for the first time, and the computation with uninitialized values also makes some computations in the 'forward' directions wrong.* (If the math calculations started with random initial values, one can hardly notice the effect.) This acceleration means a higher operating speed, but given that it slows down the mathematical convergence of the operation, it needs additional iterations. As a consequence of using this method, the computation gets considerably faster (and at the same time uses much more power), but it does not deliver the mathematically expected result, given that *the time windows overlap; thus, the system does not fulfill von Neumann's "proper sequencing" requirement*. For this reason, several limitations and disadvantages of "deep learning" have been published. [143,144]

With the introduction of recurrency, the case turns even worse. As [141] tells explicitly, *the recursive method of deep layer learning corresponds to using a very high (theoretically, infinite) number of linear layers with identical weights*, even in the case of a low number of layers (see [141], Fig. 2). One single operation starts an infinitely repeated operation (and all repetitions start another infinitely repeated operation). In a strict sense, a started operation would never terminate. *The time window corresponding to the "deep learning" operation becomes infinitely wide*, so the designer must forcefully terminate the computation time window to reach a reasonable operation speed. Due to trimming, the result is not deterministic.

The mathematical method alone explains why the training is such a slow procedure: even the correct training method can adjust its weights painfully slowly. (Ironically enough, one can experience learning despite the identical weights because the delays caused by the bus operation intermix the messages with fixed weights in the virtual layers and those with adjusted weights in the actual layers. Three steps forward, two steps backward; spiced using uninitialized parameters.) This explanation means that from the point of view of learning, the recurrent way of operation is the possible worst solution: the result of "training" manifests the slower, the larger the system, given that the infinite number of virtual layers shows genuine communication needs. The operation is not singular (although it provokes the feared "stack overflow" error if implemented as a subroutine) only thanks to the finite execution time (prolonged transfer time, typically through the bus): the new computation need cannot block the operation continuously. Given that there is no "End Computing" signal, the "computational time window" cannot be interpreted, so *it is unclear what is delivered as a result at any time. Von Neumann and Turing would say that the trimmed series computation could not be accurate.*

Unfortunately, mathematics rediscovered "Zeno's Paradox of the Tortoise and Achilles," in which a simple operation is replaced by summing up an infinite sum. The manufacturers of computing electronics implement learning in the latter form, maximizing computing's inefficiency. Essentially, this is why "*machine learning is on track to consume all the energy being supplied, a model that is costly, inefficient, and unsustainable*"[145].

#### 5.4.2. Training Anns

One of the most shocking features (for a recent review of limitations, see [144]) of ANN is their several weeks-long training time (several months for the new, unreasonably enormous systems, with their over hundred billion parameters [87]), even for (compared to the functionality of the brain) simple tasks. The mathematical methods do not comprise time dependence; however, the technical implementation of ANNs does. As their time-dependence is discussed in [97], the delivery times of new neuronal outputs (simultaneously serving as new neuronal inputs) are only loosely coupled. The assumption that one can produce an output and an input simultaneously works only in the timeless "classic computing"; see the discussion of serial bus's temporal behavior [97].

To comprehend the effect of considering temporal behavior, see the temporal diagram of a 1-bit adder in Figure 3. The system has a fixed time to read the result when adding adders. The time is set up so that all bits are relaxed and receive the final result only; the adder is synchronized.

However, the case of ANNs is different. In the ANN, the bus delivers the signals, and the interconnection type and sequence depend on many factors (ranging from the kind of task to the actual inputs). During training ANNs, their feedback complicates the case. The only fixed aspect of timing is that a neuronal input arrives inevitably only after a partner produces it. However, the time ordering of delivered events may differ from the assumed one. It depends on the technical parameters of delivery rather than the logic that generates them.

When using time stamping, there are two bad choices. Option one is for neurons to have a (biological) sending time-ordered input queue and begin processing when all partner neurons have sent their message. That needs a synchrony signal, leading to severe performance loss. Option two is processing the messages as soon as they arrive, enabling us to give feedback to a neuron when processing a message with a timestamp referring to a biologically earlier time but received physically later. In all cases, it is worth considering if their effect exceeds some tolerance level compared to the last state. Moreover, one shall mitigate the need for communication also in this way.

When training ANNs, one starts showing an input, and the system begins to work. It uses its synaptic weights valid before showing that input. Its initial weights may be randomized or correspond to the previous input data. The system sends correct signals, but a receiver processes a signal only after it is physically delivered (whether or not the message envelope contains a timestamp). It may start to adjust its weight to a state that is not yet defined. In Figure 3, the indefinite time of the first AND gate is relatively short, while the OR has a long indefinite time.

When playing chess against an opponent, one can use advantageously a faster computer to analyze the forthcoming moves. It can even compute all possible future moves before its opponent makes the next move. However, before publishing its next move, it must wait for its opponent's next move; otherwise, it may publish a wrong move. When it sends feedback to its opponent as soon as the next move is computed–i.e., without synchronization–it results in a quickly computed but possibly wrong move. Without synchronization, the faster the computer, the worse its performance as a chess player. With synchronization, the faster the computer, the higher its idle activity.

At the beginning of their operation, some component neurons of the network may have undefined states and weights. In their operation (essentially an iteration), without synchronization, *the actors, in most cases, use wrong input signals, and they surely adjust their weights to false signals initially and with significant time delay at later stages*. In a lucky case (considering that more complex systems are working with unstable states), the system will converge but painfully slowly. Or not at all. *Neglecting networks' temporal behavior leads to painfully slow and doubtful convergence.*

*Synchronization is a must, even in ANNs. Care must taken when using accelerators, feedback, and recurrent networks. The time matters.* To provide faster feedback, computing neuronal results faster cannot help. *The feedback that was received delivers the state variables that were valid long ago.* In biology, spiking is also a "look at me" signal: the feedback shall be addressed to *that* neuron that caused the change (see the Hebbian learning). Without considering the spiking time (organizing neuron polling in a software cycle), neurons receive feedback about "the effect of all fellow neurons, including me". Receiving a spike defines the beginning of its signal validity; "leaking" also defines its "expiration time". *Their temporal behavior is vital for spiking networks.*

One must drop some result/feedback events because of long queuing to provide seemingly higher performance in excessive systems. The feedback the neuron receives provides a logical dependence that the physical implementation of the computing system converts to temporal dependence [97]. The feedback arrives later, so those messages stand at the end of the queue. Because of this, it is highly probable that they "*are dropped if the receiving process is busy over several delivery cycles*" [14]. In excessive systems, undefined inputs may establish the feedback. In addition, the system may neglect (maybe correct) feedback. Another danger when simulating an asynchronous system on a system that is using at least one single centrally synchronized component introduces a hidden "clock signal" [32], which degrades the system's computing efficiency by orders of magnitude.

[105] provides an excellent "experimental proof" of the claims above: "*Yet the task of training such networks remains a challenging optimization problem. Several related problems arise. Very long training time (several weeks on modern computers, for some problems), the potential for over-fitting (whereby the learned function is too specific to the training data and generalizes poorly to unseen data), and more technically, the vanishing gradient problem*". "*The immediate effect of activating fewer units is that propagating information through the network will be faster, both at training and test time.*" The intention to compute feedback faster has its price: "*As $\lambda_s$ increases, the running time decreases, but so does performance.*" Introducing the spatiotemporal behavior of ANNs improved the efficacy of video analysis significantly [146]. Investigations in the time domain directly confirmed the role of time (mismatching): "*The CNN models are more sensitive to low-frequency channels than high-frequency channels*" [147]. The feedback can follow slow changes with less difficulty than faster changes.

## 6. Tendency of Computing Performance

### *6.1. Energy Consumption*

It is a common fallacy that the brain needs 20$W$ for its operation. According to Levy's measurements, it is about 0.1$W$[46], making the efficiency comparison to technical computing even more favorable. Another significant aspect is that "communication consumes 35 times more energy than computation in the human cortex". Their result also suggests that *the role of transferring data must not be neglected in neural systems, neither from the point of view of the needed power nor from the point of view of the time needed to carry out the operation*. An interesting parallel is that, in electronic computing [148], the same ratio was about 5, mainly because of the increased power consumption of processors due to the increased dispersion[21] and the data delivery within the processor. Von Neumann's principle "execute machine instructions one after the other until halted" with the introducing I/O and "idle" cycle of operating systems, leads to less than half of the energy being used [87] for computing "run from the beginning to the end of the computation", the rest is needed for the infrastructure. The dedicated power consumption measurements showed [87] that only 2% of the energy is used by the CPU, 22.7% by the RAM, and 75.3% by the GPU. In another grouping, 77.3% is used for computing and 22.7% for storing data. In the pre-GPU age, about 20% was used for computing, 80% for data handling [148]; *the ratio has reversed within a decade, despite the technological advancement, thanks to the GPU and AI that wastes computing power* by using it inefficiently.

Unfortunately, in deep learning, mathematics re-discovered "Zeno's Paradox of the Tortoise and Achilles," in which a simple operation is replaced by recursively summing up an infinite sum. The manufacturers of computing electronics implement learning in the latter form, maximizing computing's inefficiency. Essentially, this is why "*machine learning is on track to consume all the energy being supplied, a model that is costly, inefficient, and unsustainable*"[145].

### 6.2. Computing Efficiency

In the historically first vast computing systems (supercomputers), it was early noticed that the performance of the parallelized sequential systems strongly depended on the type of system's workload. When running "real-life" programs (the benchmark HPCG was standardized to imitate the that-time-typical workload), as opposed to the "racing only" benchmark HPL, well below one percent of the consumed energy goes for computing. This proportion is getting much worse if conventional systems attempt to mimic biology [86]. Their operating principle delivers inherent limitations [54], manifesting in neural network environments more intensely due to hardware and software reasons.

In the race for producing higher performance numbers, the number of processors has risen to several million, even though those vast systems can use only about 10% of their cores for "real-life" tasks. Their increasing utilization for "big data" problems, for more sophisticated simulations, and especially for AI-related problems re-discovered that the "the larger, the better" principle is wrong. *The absolute performance of computing systems has a maximum performance, and increasing the system's size above that value increases power consumption but decreases absolute computing performance.* The phenomenon has been known since the classic work [83]. For their task, for bio-informatical applications [84], for brain simulation [32,86], only a few dozen of processors can be used reasonably. Similarly, artificial neural networks have computational performance limitation rooflines, as pointed out both experimentally and theoretically [85,86,149]. Attempting to exceed that roofline is hopeless, including using deep learning, which leads to horrific power consumption.

Von Neumann's principle "execute machine instructions one after the other until halted" with the introducing I/O and "idle" cycle of operating systems, leads to less than half of the energy being used [87] for computing "run from the beginning to the end of the computation", the rest is needed for the infrastructure.

For the first reading, it seems shocking and unbelievable that "Substantial improvements in computing energy efficiency, by up to ten orders of magnitude, will be required to solve major computing problems" [150]. A simple calculation can estimate the energy and computing efficiency of deep learning. If we consider the characteristic operating time of a biological network is 1 *msec* and it learns some activity in 10 *sec*, the ratio of learning time to elementary operation is $10^4$. Given that the characteristic operating time of an electronic artificial network is \$1 *ns* and it learns the same activity in nearly two weeks ($10^6$ *sec*), the same ratio is $10^{15}$. The gigantic new LMM system needs 118 days of training [87]. The efficiency of a processor implementation~[87] is about \$$10^{-3}$, and also $10^{-3}$ for the efficiency of the implementation of the software stack [76]; in this way, one arrives at the result that the efficiency of deep learning is about $10^{-6} \ldots 10^{-4}$. *Neglecting von Neumann's principles led to such a catastrophic loss in computing efficiency.*

## 7. Conclusions

The stalled performance of single-processors, parellelized sequential computing and neuronal networks, topped by the enormously increased computing demand casused by the appearance of AI brought to light that computing efficiency is desperately low. The development of manufacturing technology (including discovering new materials and effects) successfully counterbalanced, for decades, the decreasing computing and energetic efficiency. The explosive need for computing power made evident that the theoretical basis of computing and the principles of technological solutions must also be changed. They are outdated; instead of incremental development, rethinking from the ground up and leap-like development is needed to enter the "next level" [75,151]. Improving computing at some spots, including some grasped-out biological solutions without context, is insufficient, and developing low-power technology alone cannot solve the problem. To solve the large-scale computing efforts we face, we need a re-discovered computing; first of all theoretically and also in its technical details. We discussed some of the problems accumulated during the past decades and proposed collecting and applying the already existing solutions. We need a 'deep understanding' instead of 'deep learning'.

## Glossary

| | |
|---|---|
| ACM | Association for Computing Machinery |
| ANN | Artificial Neural Newtwork |
| AI | Artificial Intelligence |
| AIS | Axon Initial Segment |
| AP | Action Potential |
| APTD | Action Potential Time Derivative |
| ChatGPT | ChatGPT |
| CNN | Computer Neural Network |
| CPU | Central Processing Unit |
| CSTB | Computer Science and Telecommunications Board |
| EM | electromagnetic |
| FPGA | Field Programmable Gate Array |
| GPU | Graphic Processing Unit |
| HPC | High Performance Computing |
| HPL | High Performance Linpack |
| HPCG | High-Performance Conjugate Gradients |
| HT | hyper-thread |
| HW | hardware |
| I/O | Input/Output |
| ISA | Instruction Set Architecture |
| ISI | Inter-Spike Interval |
| LLM | Large Language Model |
| MCP | Multi-Core Processor |
| ML | Machine Learning |
| OPS | Operations Per Second |
| OS | Operating System |
| PFS | Precise Firing Sequence |
| PU | Processing Unit |
| RC | Reconfigurable Computing |
| SPA | Single Processor Approach |
| PSP | Post-Synaptic Potential |
| SNN | Spiking Neural Network |
| SIMDA | Single Instruction Multiple Data |
| SOPS | Synaptic Operations Per Second |
| SW | software |

## References

1. Ngai, J. BRAIN @ 10: A decade of innovation. *Neuron* **2024**, *112*. https://doi.org/10.1016/j.neuron.2024.09.007.
2. Johnson, D.H., Information theory and neuroscience: Why is the intersection so small? In *2008 IEEE Information Theory Workshop*; IEEE, 2008; pp. 104–108. https://doi.org/10.1109/ITW.2008.4578631.
3. Human Brain Project, E. Human Brain Project. https://www.humanbrainproject.eu/en/, 2018.

4. Chu, D.; Prokopenko, M.; Ray, J.C. Computation by natural systems. https://www.researchgate.net/publication/328398755_Computation_by_natural_systems, 2018. Accessed: 2024-03-30, https://doi.org/10.1098/rsfs.2018.0058.
5. Almog, M.; Korngreen, A. Is realistic neuronal modeling realistic? *J Neurophysiol.* **2016**, *5*, 2180–2209. https://doi.org/doi:10.1152/jn.00360.2016.
6. Shannon, C.E. A mathematical theory of communication. *The Bell System Technical Journal* **1948**, *27*, 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.
7. Végh, J.; Berki, Á.J. Towards generalizing the information theory for neural communication. *Entropy* **2022**, *24*, 1086–. https://doi.org/10.3390/e24081086.
8. Shannon, C.E. The Bandwagon. *IRE Transactions in Information Theory* **1956**, *2*, 3.
9. Nizami, L. Information theory is abused in neuroscience. *Cybernetics & Human Knowing* **2019**, *26*, 47–97.
10. Brette, R. Is coding a relevant metaphor for the brain? *The Behavioral and brain sciences* **2018**, *42*, e215. https://doi.org/10.1017/S0140525X19000049.
11. Schuman, C.D.; Potok, T.E.; Patton, R.M.; Birdwell, J.D.; Dean, M.E.; Rose, G.S.; Plank, J.S. A Survey of Neuromorphic Computing and Neural Networks in Hardware. https://arxiv.org/abs/1705.06963 (Accessed on Sep 10, 2022), 2017.
12. Kandel, E.R.; Schwartz, J.H.; Jessell, T.M.; abd A. J. Hudspeth, S.A.S. *Principles of Neural Science*, 5 ed.; The McGraw-Hill, 2013.
13. Young, A.R.; Dean, M.E.; Plank, J.S.; S. Rose, G. A Review of Spiking Neuromorphic Hardware Communication Systems. *IEEE Access* **2019**, *7*, 135606–135620. https://doi.org/10.1109/ACCESS.2019.2941772.
14. van Albada, S.J.; Rowley, A.G.; Senk, J.; Hopkins, M.; Schmidt, M.; Stokes, A.B.; Lester, D.R.; Diesmann, M.; Furber, S.B. Performance Comparison of the Digital Neuromorphic Hardware SpiNNaker and the Neural Network Simulation Software NEST for a Full-Scale Cortical Microcircuit Model. *Frontiers in Neuroscience* **2018**, *12*, 291.
15. Moradi, S.; Manohar, R. The impact of on-chip communication on memory technologies for neuromorphic systems. *Journal of Physics D: Applied Physics* **2018**, *52*, 014003.
16. Carbone, J.N.; Crowder, J.A. THE GREAT MIGRATION: INFORMATION CONTENT TO KNOWLEDGE USING COGNITION BASED FRAMEWORKS. In Proceedings of the Biomedical Engineering; Suh, S.C.; Gurupur, V.P.; Tanik, M.M., Eds., New York, NY, 2011; pp. 17–46.
17. Europe spent €600 million to recreate the human brain in a computer. How did it go? *Nature* **2023**, *620*, 718–720. https://doi.org/10.1038/d41586-023-02600-x.
18. Végh, J. von Neumann's missing "Second Draft": what it should contain. In Proceedings of the Proceedings of the 2020 International Conference on Computational Science and Computational Intelligence (CSCI'20: December 16-18, 2020, Las Vegas, Nevada, USA. IEEE Computer Society, 2020, pp. 1260–1264. https://doi.org/10.1109/CSCI51800.2020.00235.
19. Backus, J. Can Programming Languages Be liberated from the von Neumann Style? A Functional Style and its Algebra of Programs. *Communications of the ACM* **1978**, *21*, 613–641.
20. János Végh. Why does von Neumann obstruct deep learning? In Proceedings of the 2023 IEEE 23rd International Symposium on Computational Intelligence and Informatics (CINTI), Budapest, Hungary, 2023, pp. 000165–000170. https://doi.org/10.1109/CINTI59972.2023.10382120.
21. Végh, J. Revising the Classic Computing Paradigm and Its Technological Implementations. *Informatics* **2021**, *8*. https://doi.org/10.3390/informatics8040071.
22. Waser, R., Ed. *Advanced Electronics Materials and Novel Devices*; Nanoelectronics and Information Technology, Wiley-VCH, 2012.
23. Buzsáki, G. Neural syntax: cell assemblies, synapsembles, and readers. *Neuron* **2010**, *68*, 362–85. https://doi.org/10.1016/j.neuron.2010.09.023.
24. Buzsáki, G.; Mizuseki, K. The log-dynamic brain: how skewed distributions affect network operations. *Nature Reviews Neuroscience* **2014**, *15*, 264–278. https://doi.org/10.1038/nrn3687.
25. Caporale, N.; Dan, Y. Spike Timing–Dependent Plasticity: A Hebbian Learning Rule. *Annual Review of Neuroscience* **2008**, *31*, 25–46. PMID: 18275283, https://doi.org/10.1146/annurev.neuro.31.060407.125639.
26. Madl T, Baars BJ, F.S. The timing of the cognitive cycle. *PLoS One* **2011**, *6*. https://doi.org/10.1371/journal.pone.0014803.
27. Cai, M.; Demmans Epp, C. Exploring the Optimal Time Window for Predicting Cognitive Load Using Physiological Sensor Data **2024**. https://doi.org/10.48550/arXiv.2406.13793.

28. Linder, B.; Garcia-Ojalvo, J.; Neiman, A.; Schimansky-Geier, L. Effects of noise in excitable systems. *Physics reports* **2004**, *392*, 321–424.
29. Perkel, D.; B., M. Electrotonic properties of neurons: steady-state compartmental model. *J Neurophysiol.* **1978**, *41*, 621–39. https://doi.org/10.1152/jn.1978.41.3.621.
30. Smirnova, L.e.a. Organoid intelligence (OI): the new frontier in biocomputing and intelligence-in-a-dish. *Frontiers in Science* **2023**. https://doi.org/10.3389/fsci.2023.1017235.
31. Kunkel, S.; Schmidt, M.; Eppler, J.M.; Plesser, H.E.; Masumoto, G.; Igarashi, J.; Ishii, S.; Fukai, T.; Morrison, A.; Diesmann, M.; et al. Spiking network simulation code for petascale computers. *Frontiers in Neuroinformatics* **2014**, *8*, 78. https://doi.org/10.3389/fninf.2014.00078.
32. Végh, J. How Amdahl's Law limits performance of large artificial neural networks. *Brain Informatics* **2019**, *6*, 1–11. https://doi.org/10.1186/s40708-019-0097-2.
33. US DOE Office of Science. Report of a Roundtable Convened to Consider Neuromorphic Computing Basic Research Needs. https://science.osti.gov/-/media/ascr/pdf/programdocuments/docs/Neuromorphic-Computing-Report_FNLBLP.pdf, 2015.
34. Markovic, D.; Mizrahi, A.; Querlioz, D.; Grollier, J. Physics for neuromorphic computing. *Nature Reviews Physics* **2020**, *2*, 499–510. https://doi.org/https://www.nature.com/articles/s42254-020-0208-2.pdf.
35. Mehonic, A.; Kenyon, A.J. Brain-inspired computing needs a master plan. *Nature* **2022**, *604*, 255–260. https://doi.org/10.1038/s41586-021-04362-w.
36. von Neumann, John. *John von Neumann and the Origins of Modern Computing*; Yale University Press, 2012.
37. Koch, C. *Biophysics of Computation*; Oxford University Press: New York, Oxford, 1999.
38. Abbott, L.; Sejnowski, T.J. *Neural Codes and Distributed Representations*; Cambridge, MA: MIT Press, 1999.
39. Lytton, W.W. *From Computer to Brain*; Springer, 2002.
40. Sejnowski, T.J. The Computer and the Brain Revisited. *IEEE Annals of the History of Computing* **1989**, *11*, 197–201. https://doi.org/10.1109/MAHC.1989.10028.
41. Tsafrir, D. The Context-switch Overhead Inflicted by Hardware Interrupts (and the Enigma of Do-nothing Loops). In Proceedings of the Proceedings of the 2007 Workshop on Experimental Computer Science, San Diego, California, New York, NY, USA, 2007; ExpCS '07, pp. 3–3.
42. David, F.M.; Carlyle, J.C.; Campbell, R.H. Context Switch Overheads for Linux on ARM Platforms. In Proceedings of the Proceedings of the 2007 Workshop on Experimental Computer Science, San Diego, California, New York, NY, USA, 2007; ExpCS '07. https://doi.org/10.1145/1281700.1281703.
43. nextplatform.com. CRAY revamps clusterstor for the exascale era. https://www.nextplatform.com/2019/10/30/cray-revamps-clusterstor-for-the-exascale-era/, 2019. Accessed: 2023-09-10.
44. Kendall, J.D.; Kumar, S. The building blocks of a brain-inspired computer. *Appl. Phys. Rev.* **2020**, *7*, 011305. https://doi.org/10.1063/1.5129306.
45. Végh, J.; Berki, Á.J. On the Role of Speed in Technological and Biological Information Transfer for Computations. *Acta Biotheoretica* **2022**, *70*, 26. https://doi.org/10.1007/s10441-022-09450-6.
46. Berger, T.; Levy, W.B. A Mathematical Theory of Energy Efficient Neural Computation and Communication. *IEEE Transactions on Information Theory* **2010**, *56*, 852–874. https://doi.org/10.1109/TIT.2009.2037089.
47. Schrödinger, E., IS LIFE BASED ON THE LAWS OF PHYSICS? In *What is Life?: With Mind and Matter and Autobiographical Sketches*; Canto, Cambridge University Press, 1992; p. 76–85.
48. Végh, J. The non-ordinary laws of physics describing life. *Foundations of Physics* **2025**, *1*.
49. Quirion, R. Brain organoids: are they for real? https://www.frontiersin.org/journals/science/articles/10.3389/fsci.2023.1148127/full, 2023. https://doi.org/10.3389/fsci.2023.1148127.
50. von Neumann, J. First draft of a report on the EDVAC. *IEEE Annals of the History of Computing* **1993**, *15*, 27–75. https://doi.org/10.1109/85.238389.
51. Bell, G.; Bailey, D.H.; Dongarra, J.; Karp, A.H.; Walsh, K. A look back on 30 years of the Gordon Bell Prize. *The International Journal of High Performance Computing Applications* **2017**, *31*, 469–484.
52. IEEE. IEEE Rebooting Computing. http://rebootingcomputing.ieee.org/, 2013.
53. Cadareanu, P.; Reddy C, N.; Almudever, C.G.; Khanna, A.; Raychowdhury, A.; Datta, S.; Bertels, K.; Narayanan, V.; Ventra, M.D.; Gaillardon, P.E. Rebooting Our Computing Models. In Proceedings of the 2019 Design, Automation Test in Europe Conference Exhibition (DATE) Florence, Italy; 25-29 March 2019, 2019, pp. 1469–1476. https://doi.org/10.23919/DATE.2019.8715167.
54. Végh, J. Finally, how many efficiencies the supercomputers have? *The Journal of Supercomputing* **2020**, *76*, 9430–9455, regularly updated at https://arxiv.org/abs/2001.01266.

55. Levy, W.B.; Calvert, V.G. Communication consumes 35 times more energy than computation in the human cortex, but both costs are needed to predict synapse number. *Proceedings of the National Academy of Sciences* **2021**, *118*, e2008173118. https://doi.org/10.1073/pnas.2008173118.
56. Hameed, R.; Qadeer, W.; Wachs, M.; Azizi, O.; Solomatnikov, A.; Lee, B.C.; Richardson, S.; Kozyrakis, C.; Horowitz, M. Understanding Sources of Inefficiency in General-purpose Chips. In Proceedings of the Proceedings of the 37th Annual International Symposium on Computer Architecture, Saint-Malo, France, New York, NY, USA, 2010; ISCA '10, pp. 37–47. https://doi.org/10.1145/1815961.1815968.
57. Végh, J. Introducing the Explicitly Many-Processor Approach. *Parallel Computing* **2018**, *75*, 28 – 40.
58. Végh, J. How to Extend Single-Processor Approach to Explicitly Many-Processor Approach. In Proceedings of the Advances in Software Engineering, Education, and e-Learning; Arabnia, H.R.; Deligiannidis, L.; Tinetti, F.G.; Tran, Q.N., Eds. Springer International Publishing, 2021, pp. 435–458.
59. Birkhoff, G. and Von Neumann, J.. The logic of quantum mechanics. *Annals of mathematics* **1936**, pp. 823–843.
60. Cho, A. Tests measure progress of quantum computers. *Science* **2018**, *364*, 1218–1219. https://doi.org/10.1126/science.364.6447.1218.
61. Ruiz-Perez, L.; Garcia-Escartin, J.C. Quantum arithmetic with the quantum Fourier transform. *Quantum Information Processing* **2017**, *16*, 152.
62. Goychuk, I.; Hänggi, P.; Vega, J.L.; Miret-Artés, S. Non-Markovian stochastic resonance: Three-state model of ion channel gating. *Phys. Rev. E* **2005**, *71*, 061906. https://doi.org/10.1103/PhysRevE.71.061906.
63. Feynman, R.P. *Feynman Lectures on Computation*; CRC Press, 2018.
64. Asanovic, K.; Bodik, R.; Demmel, J.; Keaveny, T.; Keutzer, K.; Kubiatowicz, J.; Morgan, N.; Patterson, D.; Sen, K.; Wawrzynek, J.; et al. A View of the Parallel Computing Landscape. *Comm. ACM* **2009**, *52*, 56–67.
65. Esmaeilzadeh, H.; Blem, E.; St. Amant, R.; Sankaralingam, K.; Burger, D. Dark Silicon and the End of Multicore Scaling. *IEEE Micro* **2012**, *32*, 122–134.
66. Shafique, M.; Garg, S. Computing in the dark silicon era: Current trends and research challenges. *IEEE Design and Test* **2017**, *34*, 8–23. https://doi.org/10.1109/MDAT.2016.2633408.
67. Tenhunen, M.H.H..A.M.R..P.L..A.J..A.M..C.B..H. Can Dark Silicon Be Exploited to Prolong System Lifetime? *IEEE Design and Test* **2017**, *34*, 51–59. https://doi.org/10.1109/MDAT.2016.2633408.
68. Markov, I. Limits on fundamental limits to computation. *Nature* **2014**, *512*, 147–154. http://download.nap.edu/cart/download.cgi?&record_id=12980.
69. Bourzac, K. Streching supercomputers to the limit. *Nature* **2017**, *551*, 554–556.
70. Service, R.F. Design for U.S. exascale computer takes shape. *Science* **2018**, *359*, 617–618.
71. S. Furber and S. Temple. Neural systems engineering. *J. R. Soc. Interface* **2007**, *4*, 193–206. https://doi.org/10.1098/rsif.2006.0177.
72. Wang, C.; Liang, S.J.; Wang, C.Y.; Yang, Z.Z.; Ge, Y.; Pan, C.; Shen, X.; Wei, W.; Zhao, Y.; Zhang, Z.; et al. Beyond von Neumann. *Nature Nanotechnology* **2020**, p. 507. https://doi.org/10.1038/s41565-020-0738-x.
73. J. P. Eckert, J.; Mauchly, J.W. Automatic High-Speed Computing: A Progress Report on the EDVAC. Technical Report of Work under Contract No. W-670-ORD-4926, Supplement No 4, Moore School Library, University of Pennsylvania, Philadephia, 1945.
74. Schlansker, M.; Rau, B. EPIC: Explicitly Parallel Instruction Computing. *Computer* **2000**, *33*, 37–45.
75. Fuller, S.H.; Millett, L.I. Computing Performance: Game Over or Next Level? *Computer* **2011**, *44*, 31–38.
76. Ousterhout, J.K. Why Aren't Operating Systems Getting Faster As Fast As Hardware? http://www.stanford.edu/~ouster/cgi-bin/papers/osfaster.pdf, 1990. Accessed: 2023-09-10.
77. L. Sha and R. Rajkumar and J.P. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers* **1990**, *39*, 1175–1185. https://doi.org/10.1109/12.57058.
78. Babaoglu, O.; Marzullo, K.; Schneider, F.B. A formalization of priority inversion. *Real-Time Systems* **1993**, *5*, 285–303. https://doi.org/10.1007/BF01088832.
79. Amdahl, G.M. Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. In Proceedings of the AFIPS Conference Proceedings, 1967, Vol. 30, pp. 483–485. https://doi.org/10.1145/1465482.1465560.
80. ARM. big.LITTLE technology, 2011.
81. Ao, Y.; Yang, C.; Liu, F.; Yin, W.; Jiang, L.; Sun, Q. Performance Optimization of the HPCG Benchmark on the Sunway TaihuLight Supercomputer. *ACM Trans. Archit. Code Optim.* **2018**, *15*, 11:1–11:20.
82. Gordon, S., Ed. *The Synaptic Organization of the Brain*, 5 ed.; Oxford Academic, New York, 2006. https://doi.org/10.1093/acprof:oso/9780195159560.001.1.

83. Singh, J.P.; Hennessy, J.L.; Gupta, A. Scaling Parallel Programs for Multiprocessors: Methodology and Examples. *Computer* **1993**, *26*, 42–50. https://doi.org/10.1109/MC.1993.274941.
84. D'Angelo, G.; Rampone, S. Towards a HPC-oriented parallel implementation of a learning algorithm for bioinformatics applications. *BMC Bioinformatics* **2014**, *15*.
85. Keuper, J.; Pfreundt, F.J. Distributed Training of Deep Neural Networks: Theoretical and Practical Limits of Parallel Scalability. In Proceedings of the 2nd Workshop on Machine Learning in HPC Environments (MLHPC). IEEE, 2016, pp. 1469–1476. https://doi.org/10.1109/MLHPC.2016.006.
86. Végh, J. Which scaling rule applies to Artificial Neural Networks. *Neural Computing and Applications* **2021**. https://doi.org/10.1007/s00521-021-06456-y.
87. Luccioni, A.S.; Viguier, S.; A-N, L. Estimating the Carbon Footprint of BLOOM, a 176B Parameter Language Model. *J. Machine Learning Research* **2023**, *24*, 1–15.
88. Matheou, G.; Evripidou, P. Architectural Support for Data-Driven Execution. *ACM Trans. Archit. Code Optim.* **2015**, *11*, 52:1–52:25.
89. Denning, P.J.; Lewis, T. Exponential Laws of Computing Growth. *Communications of the ACM* **2017**, pp. 54–65.
90. Vetter, J.S.; DeBenedictis, E.P.; Conte, T.M. Architectures for the Post-Moore Era. *IEEE Micro* **2017**, *37*, 6–8.
91. Nature. In AI, is bigger always better? *Nature* **2023**, *615*. Accessed: 2023-09-10.
92. Smith, B. Reinventing computing. In Proceedings of the International Supercomputing Conference, 2007.
93. Lee, V.W.; Kim, C.; Chhugani, J.; Deisher, M.; Kim, D.; Nguyen, A.D.; Satish, N.; Smelyanskiy, M.; Chennupaty, S.; Hammarlund, P.; et al. Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU. In Proceedings of the Proceedings of the 37th Annual International Symposium on Computer Architecture, New York, NY, USA, 2010; ISCA '10, Saint-Malo, France, pp. 451–460. https://doi.org/10.1145/1815961.1816021.
94. cortical.io. Third AI Winter ahead? Why OpenAI, Google et Co are heading towards a dead-end. https://www.cortical.io/blog/third-ai-winter-ahead-why-openai-google-co-are-heading-towards-a-dead-end/, 2022.
95. de Macedo Mourelle, L.; Nedjah, N.; Pessanha, F.G., Reconfigurable and Adaptive Computing: Theory and Applications; CRC press, 2016; chapter 5: Interprocess Communication via Crossbar for Shared Memory Systems-on-chip. https://doi.org/10.1201/b19157-7.
96. Beggs, J.M.; Plenz, D. Neuronal Avalanches in Neocortical Circuits. *Journal of Neuroscience* **2003**, *23*, 11167–11177, [https://www.jneurosci.org/content/23/35/11167.full.pdf]. https://doi.org/10.1523/JNEUROSCI.23-35-11167.2003.
97. Végh, J. Introducing Temporal Behavior to Computing Science. In Proceedings of the Advances in Software Engineering, Education, and e-Learning; Arabnia, H.R.; Deligiannidis, L.; Tinetti, F.G.; Tran, Q.N., Eds. Springer International Publishing, 2021, pp. 471–491.
98. Végh, J. A configurable accelerator for manycores: the Explicitly Many-Processor Approach. *ArXiv e-prints* **2016**, [1607.01643].
99. Mahlke, S.; Chen, W.; Chang, P.; Hwu, W.M. Scalar program performance on multiple-instruction-issue processors with a limited number of registers. In Proceedings of the Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences, 1992, Vol. 1, pp. 34 – 44. https://doi.org/10.1109/HICSS.1992.183141.
100. Chicca, E.; Indiveri, G. A recipe for creating ideal hybrid memristive-CMOS neuromorphic processing systems. *Applied Physics Letters* **2020**, *116*, 120501. https://doi.org/10.1063/1.5142089.
101. Strukov, D.; Indiveri, G.; Grollier, J.; Fusi, S. Building brain-inspired computing. *Nature Communications* **2019**, *10*, 4838. https://doi.org/10.1038/s41467-019-12521-x.
102. Wang, C.; Liang, S.J.; Wang, C.Y.; Yang, Z.Z.; Ge, Y.; Pan, C.; Shen, X.; Wei, W.; Zhao, Y.; Zhang, Z.; et al. Scalable massively parallel computing using continuous-time data representation in nanoscale crossbar array. *Nature Nanotechnology* **2021**. https://doi.org/10.1038/s41565-021-00943-y.
103. Pan, X.; Shi, J.; Wang, P.; Wang, S.; Pan, C.; Yu, W.; Cheng, B.; Liang, S.J.; Miao, F. Parallel perception of visual motion using light-tunable memory matrix. *Science Advances* **2023**, *9*, eadi4083. https://doi.org/10.1126/sciadv.adi4083.
104. Han, S.; Pool, J.; Tran, J.; Dally, W.J. Learning both Weights and Connections for Efficient Neural Networks. https://arxiv.org/pdf/1506.02626.pdf, 2015.
105. Bengio, E.; Bacon, P.L.; Pineau, J.; Precu, D. Conditional Computation in Neural Networks for faster models. https://arxiv.org/pdf/1511.06297, 2016, [arXiv:cs.LG/1511.06297]. Accessed: 2023-08-30.

106. Johnston, D.; sin Wu, S.M. *Foundations of Cellular Neurophysiology*; Massachusetts Institute of Technology: Cambridge, Massachusetts and London, England, 1995.
107. Végh, J.; Berki, A.J. Revisiting neural information, computing and linking capacity. *Mathematical Biology and Engineering* **2023**, *20*, 12380–12403. https://doi.org/10.3934/mbe.2023551.
108. Somjen, G. *SENSORY CODING in the mammalian nervous system*; New York, MEREDITH CORPORATION, 1972. https://doi.org/10.1007/978-1-4684-1707-4.
109. Susi, G.; Garcés, P.; Paracone, E.; Cristini, A.; Salerno, M.; Maestú, F.; Pereda, E. FNS allows efficient event-driven spiking neural network simulations based on a neuron model supporting spike latency. *Nature Scientific Reports* **2021**, *11*. https://doi.org/10.1038/s41598-021-91513-8.
110. Tschanz, J.W.; Narendra, S.; Ye, Y.; Bloechel, B.; Borkar, S.; De, V. Dynamic sleep transistor and body bias for active leakage power control of microprocessors. *IEEE Journal of Solid State Circuits* **2003**, *38*, 1838 – 1845.
111. Onen, M.; Emond, N.; Wang, B.; Zhang, D.; Ross, F.M.; Li, J.; Yildiz, B.; del Alamo, J.A. Nanosecond protonic programmable resistors for analog deep learning. *Science* **2022**, *377*, 539–543, [https://www.science.org/doi/pdf/10.1126/science.abp8064]. https://doi.org/10.1126/science.abp8064.
112. Hodgkin, A.L.; Huxley, A.F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol.* **1952**, *117*, 500–544.
113. Losonczy, A.; Magee, J. Integrative properties of radial oblique dendrites in hippocampal CA1 pyramidal neurons. *Neuron* **2006**, *50*, 291–307. https://doi.org/10.1016/j.neuron.2006.03.016.
114. Leterrier, C. The Axon Initial Segment: An Updated Viewpoint. *Journal of Neuroscience* **2018**, *38*, 2135–2145. https://doi.org/10.1523/JNEUROSCI.1922-17.2018.
115. Goikolea-Vives, A.; Stolp, H. Connecting the Neurobiology of Developmental Brain Injury: Neuronal Arborisation as a Regulator of Dysfunction and Potential Therapeutic Target. *Int J Mol Sci* **2021**, *15*. https://doi.org/10.3390/ijms22158220.
116. Hasegawa, K.; ichiro Kuwako, K. Molecular mechanisms regulating the spatial configuration of neurites. *Seminars in Cell & Developmental Biology* **2022**, *129*, 103–114. Special Issue: Molecular dissection of cognition, emotion and thought by Akira Sawa & Takeshi Sakurai / Special Issue: Emerging biology of cellular protrusions in 3D architecture by Mayu Inaba and Mark Terasaki, https://doi.org/https://doi.org/10.1016/j.semcdb.2022.02.015.
117. Végh, J. Dynamic Abstract Neural Computing with Electronic Simulation. https://jvegh.github.io/DynamicAbstractNeuralComputing/ (Accessed on Feb 6, 2025), 2025.
118. Forcella, D.; Zaanen, J.; Valentinis, D.; van der Marel, D. Electromagnetic properties of viscous charged fluids. *Phys. Rev. B* **2014**, *90*, 035143. https://doi.org/10.1103/PhysRevB.90.035143.
119. McKenna, T.; Davis, J.; Zornetzer, S. *Single Neuron Computation*; Neural Networks: Foundations to Applications, Academic Press, 2014.
120. Huang, C.Y.M.; Rasband, M.N. Axon initial segments: structure, function, and disease. *Annals of the New York Academy of Sciences* **2018**, *1420*. https://doi.org/10.1111/nyas.13718.
121. Alonso1, L.M.; Magnasco, M.O. Complex spatiotemporal behavior and coherent excitations in critically-coupled chains of neural circuits. *Chaos: An Interdisciplinary Journal of Nonlinear Science* **2018**, *28*, 093102. https://doi.org/doi:10.1063/1.5011766.
122. Li, M.; Tsien, J.Z. Neural Code-Neural Self-information Theory on How Cell-Assembly Code Rises from Spike Time and Neuronal Variability. *Frontiers in Cellular Neuroscience* **2017**, *11*. https://doi.org/10.3389/fncel.2017.00236.
123. Li, M.; Tsien, J. Neural Code-Neural Self-information Theory on How Cell-Assembly Code Rises from Spike Time and Neuronal Variability. *Frontiers in Cell Neuroscience* **2017**. https://doi.org/10.3389/fncel.2017.00236.
124. D'Angelo, G.; Palmieri, F. Network traffic classification using deep convolutional recurrent autoencoder neural networks for spatial–temporal features extraction. *Journal of Network and Computer Applications* **2021**, *173*, 102890. https://doi.org/10.1016/j.jnca.2020.102890.
125. TOP500. Top500 list of supercomputers. https://www.top500.org/lists/top500/ (Accessed on Oct 24, 2021), 2021.
126. Aspray, W., John von Neumann and the Origins of Modern Computing. In *John von Neumann and the Origins of Modern Computing*; Cohen, B.; Aspray, W., Eds.; MIT Press, Cambridge, 1990; pp. 34–48.
127. Sterling, P.; Laughlin, S. *Principles of Neural Design*, 1 ed.; The MIT Press: Cambridge, Massachusetts and London, England, 2017.
128. Antle, M. C. and Silver, R.. Orchestrating time: arrangements of the brain circadian clock. *Trends Neurosci.* **2005**, *28*, 145–151.

129. Végh, J.; Ádám József Berki. Storing and Processing Information in Technological and Biological Computing Systems. In Proceedings of the The 2021 international conference on computational science and computational intelligence; foundations of computer science FCS. IEEE, 2021, Vol. 21, p. FCS4378.
130. Stone, J.V. *Principles of Neural Information Theory*; Sebtel Press, Sheffield, UK, 2018.
131. McKenzie, S.; Huszár, R.; English, D.F.; Kim, K.; Yoon, E.; Buzsáki, G. Preexisting hippocampal network dynamics constrain optogenetically induced place fields. *Neuron* **2021**, *109*. https://doi.org/10.1101/803577.
132. Jordan, M.I. Artificial Intelligence—The Revolution Hasn't Happened Yet. https://hdsr.mitpress.mit.edu/pub/wot7mkc1/release/10, 2019.
133. Science. Core progress in AI has stalled in some fields. *Science* **2020**, *368*, 6494/927. https://doi.org/10.1126/science.368.6494.927.
134. Rouleau, N.; Levin, M. Discussions of machine versus living intelligence need more clarity. *Nat Mach Intell* **2024**, *6*, 1424–1426. https://doi.org/10.1038/s42256-024-00955-y.
135. Editorial. Seeking clarity rather than strong opinions on intelligence. *Nat Mach Intell* **2024**, *6*. https://doi.org/10.1038/s42256-024-00968-7.
136. Végh, J.; Berki, A.J. Why learning and machine learning are different. *Advances in Artificial Intelligence and Machine Learning* **2021**, *1*, 131–148. https://doi.org/10.54364/AAIML.2021.1109.
137. Luk, W. Imperial College London, textbook. http://www.imperial.ac.uk/~wl/teachlocal/cuscomp/notes/chapter2.pdf (Accessed on Dec 14, 2020), 2019.
138. Black, C.D.; Donovan, J.; Bunton, B.; Keist, A. *SystemC: From the Ground Up*, second ed.; Springer: New York, 2010.
139. IEEE/Accellera. Systems initiative. http://www.accellera.org/downloads/standards/systemc, 2017.
140. Mitra, P. Fitting elephants in modern machine learning by statistically consistent interpolation. *Nature Machine Intelligence* **2021**, *3*. https://doi.org/10.1038/s42256-021-00345-8.
141. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. https://doi.org/10.1038/nature14539.
142. axios.com. Artificial intelligence pioneer says we need to start over. https://www.axios.com/2017/12/15/artificial-intelligence-pioneer-says-we-need-to-start-over-1513305524.
143. Marcus, G. Deep Learning: A Critical Appraisal. https://arxiv.org/ftp/arxiv/papers/1801/1801.00631.pdf.
144. Cremer, C.Z. Deep limitations? Examining expert disagreement over deep learning. https://doi.org/10.1007/s13748-021-00239-1, 2021. https://doi.org/10.1007/s13748-021-00239-1.
145. semiengineering.com. AI Power Consumption Exploding. https://semiengineering.com/ai-power-consumption-exploding/, 2022. Accessed: 2023-09-10.
146. Xie, S.; Sun, C.; Huang, J.; Tu, Z.; Murphy, K. Rethinking Spatiotemporal Feature Learning: Speed-Accuracy Trade-offs in Video Classification. In Proceedings of the Computer Vision – ECCV 2018; Ferrari, V.; Hebert, M.; Sminchisescu, C.; Weiss, Y., Eds., Cham, 2018; pp. 318–335.
147. Xu, K.; Qin, M.; Sun, F.; Wang, Y.; Chen, Y.K.; Ren, F. Learning in the Frequency Domain. https://arxiv.org/abs/2002.12416, 2020. Accessed: 2025-02-08, https://doi.org/10.1109/CVPR42600.2020.00181.
148. Simon, H. Why we need Exascale and why we won't get there by 2020. https://www.researchgate.net/publication/261879110_Why_we_need_Exascale_and_why_we_won't_get_there_by_2020, 2014. Accessed: 2023-09-10.
149. János Végh. How Science and Technology Limit the Performance of AI Networks. In Proceedings of the 5th International Conference on Advances in Signal Processing and Artificial Intelligence (ASPAI' 2023), 7-9 June 2023, Tenerife (Canary Islands), Spain . International Frequency Sensor Association (IFSA) Publishing, 2023, pp. 90–92.
150. nature.com. Solving the big computing problems in the twenty-first century. https://www.nature.com/articles/s41928-023-00985-1.epdf, 2023. Accessed: 2023-09-10.
151. Fuller, S.H.; Millett, L.I., The Future of Computing Performance: Game Over or Next Level?; National Academies Press, Washington, 2011. https://doi.org/10.17226/12980.