Human Systems Engineering and Design (IHSED 2024), Vol. 158, 2024, 200–210
https://doi.org/10.54941/ahfe1005543

Making Artificial Intelligence Intelligent – Solving the Control Problem for Artificial Neural Networks by Empirical Methods

Thomas Fehlmann[1] and Eberhard Kranich[2]

[1]Euro Project Office, Zurich, Switzerland
[2]Euro Project Office, Duisburg, Germany

ABSTRACT

The Graph Model of Combinatory Logic with its elements, the Combinators, is an algebraic representation of neural networks, both for natural and artificial nets. Solving the Control Problem leads to intelligent behavior in the sense of teaching, learning, and conceptualization. This requires the construction of specific fixed-point combinators that implement feedback loops based on empirical sensing. This paper explains the role of the control problem, how to solve it and how to algebraically construct these fixed-point combinators. It proposes a blueprint and technical design for intelligent AI-supported systems that can teach themselves new skills. This is an alternative to deep learning, which requires large training sets.

Keywords: Skill localization, Combinatory logic, The graph model, Neural networks, Artificial intelligence, Control problem, Empirical software programming

INTRODUCTION

Artificial Intelligence (AI) has been around for more than hundred years, initially by concepts only, but humans have been dreaming of intelligent machines and artificial beings since its beginning. One of the first mentions of mechanical robots equipped with νόος (English: nous, reasonably behaving autonomous robots of the Greek god Hephaistos) appears in the Iliad (Homer, 1898). Nevertheless, the current hype – the third of its kind in 50 years – is around software that can make use of big data and chats a lot about everything but understands nothing. The advent of *Large Language Models* (LLM) (Wolfram, 2023) made this hype possible, although its use – outside of management reports, probably – is quite limited to homework cheating, arts creation, production of fake images and spreading fear among employees who are not sure whether their work is valuable enough to withstand job replacement by some AI-enabled tool.

However, the 2012 breakthrough made it possible to solve large sparse matrices of neural networks using parallel processing (Hinton, 2012). This made a difference. For the first time, glimpses of real intelligence have been observed in very large LLM, suggesting that neural networks can exhibit intelligence indeed, and that might even hold for biological neural networks where such rather rare events have enabled an incredible growth of technology and science especially in the last few centuries.

A SHORT HISTORY OF ARTIFICIAL INTELLIGENCE

State of the Third AI-Hype

While the first AI hype was largely focused around the first perceptrons and attempts to understand natural language (Winograd, 1972) by transforming language into grammar trees, the second hype occurred around 20 years later when expert systems promised to ease access to specialist knowledge considerably. Expert systems exploit some knowledge base and provide sophisticated knowledge retrieval functionality.

Until the arrival of big data, or *Data Science*, the collection of data proved to be a major hurdle for commercial success. Notable exceptions include the search engines such as Google (Langville & Meyer., 2006) for searching the Internet and maybe Wolfram|Alpha for mathematical and scientific knowledge (Wolfram, 2023).

Nevertheless, advances in neuroscience and computing power triggered the third AI hype in the early 2010s. Again, neural networks became the focus of interest, and as their size increased by factors, they proved capable of taking on several traditional programming tasks. This is because translation among different languages is among the key capabilities of LLMs. Why should an LLM not be able to translate a system requirements specification into code? This is because detailed and meaningful system requirements specifications are among human dreams that so far never materialized, same as with fully autonomous cars.

Since equation (1) needs a finite set on the left-hand side, most probably it is a consequence of the undecidability of predicate logic (Gödel, 1931) that meaningful requirements specification remains a dream, forever.

From Dream to Reality

In 1920, after the shocking discovery that first-order logic remains undecidable (Gödel, 1931), there were attempts to fix the problem, mostly by limiting or forbidding for-all quantifiers. The idea is great when we remember the incredible suffering and damage that such quantifiers have caused in society and politics, whether through racism or prejudice. Curry et al. (1972) invented around 1930 *Combinatory Logic*. By avoiding the quantifiers of traditional predicate logic, it serves as the theoretical basis of mathematics and, above all, computer science. Fifty years later, Engeler and Scott found that a *Graph Model* exists that represents combinatory logic in an algebraic way (Engeler, 1981). The graph model consists of the powerset of *Arrow Terms*

$$\{x_1, x_2, \ldots, x_n\} \to y \tag{1}$$

representing directed graphs such as neural networks. The $x_1, x_2, \ldots, x_n$ refer to the preceding nodes of node y. Adding weights in the nodes creates an algebraic model of neural networks (Fehlmann & Kranich, Preprint, 2024). The elements of the powerset of arrow terms we call *Combinators*. Knowledge thus is represented by combinators that constitute a neural network.

Such neural networks work as a *Perceptron* (Minsky & Papert, 1972) and can be trained to recognize patterns, or as an LLM, for instance the *Generative Pretrained Translator* ChatGPT (Wolfram, 2023). We use this algebraic model to identify intelligent AI behavior.

While ChatGPT has the capability to chat only, it can be combined with a knowledge source such as Wolfram|Alpha to answer relevant questions (Wolfram, 2023). This is called *Retrieval-Augmented Generation* (RAG) (Gao, et al., 2024). It means that an LLM works together with a knowledge base. This requires that the knowledge is presented to the LLM in the form of a vector that codes for the facts. Once this interface works, adding more factual knowledge becomes cheap and simple, without requiring any new training effort for the LLM. The LLM still becomes not intelligent in the sense that it can learn how to learn, as every human or animal baby does. The term "intelligent" is misleading because the English word "intelligence" is associated with two very different concepts:

- LLMs and big data analytics are "intelligent" in the sense of the kind of intelligence that for instance pertains to organizations such as the FBI and the CIA, referring to the ability to collect relevant data;
- Nous ($\nu \acute{o} o \varsigma$) means that there is an understanding for the contextual situation, thus the ability to learn on the spot from context or from the physical environment.

Recently, signs of unexpected behavior of large neural networks, suggesting $\nu \acute{o} o \varsigma$, have been observed (Zhong et al., 2022), and Panigrahi and his colleagues at Princeton University found ways how to locate specific skills in a very large LLM (Panigrahi et al., 2023). This means that the algebraic Graph Model can be applied to artificial neural networks. Both findings are predictable by the graph model of combinatory logic. The graph model postulates the existence of *Combinators*, specific sets of arrow terms according to equation (1) that have a structural effect on neural networks by rearranging knowledge and skills according to certain rules describing calculation and reasoning. Examples of such combinators can be found in the literature (Zachos, 1978; Bimbó, 2012).

Engeler explained in his seminal paper about "how does the brain think" (Engeler, 2019) in which way the graph model can be used to algebraically construct combinators that learn and teach the biologic neural networks, the brain. This construction can be carried over to artificial nets thanks to the discovery of Panigrahi.

IMPLEMENTING COMBINATORS IN AI-ENABLED SYSTEMS

Combining Combinators

We use an analogue of Einstein's summation convention x_i to denote a set $\{x_1, x_2, \ldots, x_n\}$ (Fehlmann, 2020, p. 6). Here, $i \in \mathbb{N}$ is not simply an index, but we understand *i* as a choice function that selects the relevant preconditions from a set of possibilities. This is related to the intuitionistic variant of the axiom of choice, see (Fehlmann & Kranich, 2020).

Two combinators *M* and N, i.e., two sets of arrow terms, can be combined as follows:

$$M \cdot N \;=\; \left\{ b \middle| \exists a_i \to b \in M;\; a_i \in N \right\} \tag{2}$$

Equation (2) means, if there are combinators b_i in combinator *N* that satisfy some preconditions of *M*, then some of the right-hand combinators in *M* hold also for the combination $M \cdot N$. This yields a directed graph. This is the motivation for calling it a "Graph Model" (Fehlmann & Kranich, 2024). If the nodes carry a weight, the "existence" of an a_i is to be replaced by "exceeds the threshold value".

The alert reader will have noticed that we are using a mathematical trick here, namely inflating the arrow terms to a recursively defined set of arrow terms consisting of arrow terms.

You can base this powerset on the null set. Then, weights in nodes play no role in equation (2). If you base the powerset on some non-empty set of, say, *Observations*, then the powerset describes the behavior of artificial neural networks. As an example, we look at those used for visual recognition. The observations describe sensor input, and results of equation (2) might classify objects recognized. The response of our neural network might be used to control actuators such as a steering gear.

Combinators containing arrow terms of the form $a_i \;\to\; b$ only are called *Concepts*. Concepts can be independent from any application domain and thus can implement general principles of thought and reasoning in an *Intelligent System*. We use this term interchangeably with the term AI-enabled system. Combinators might contain both observations and concepts and have infinite set size.

In any case, the powerset is closed with respect to the application (2) and is also Turing-complete, i.e., it has the necessary structure to perform all theoretically possible calculations, see Engeler (Engeler, 1981). This finding explains why artificial neural networks have become so powerful.

The Lambda Combinator

The graph model also includes Lambda terms (Barendregt & Barendsen, 2000). *Lambda Terms* introduce a variable *x* in *M*, allowing for an application of *M* to some argument *N*

$$\lambda x.M \cdot N \tag{3}$$

In this case (3), *N* replaces all occurrences of x in M. For formal definitions, consult (Fehlmann, 2020, p. 5). In the graph model, Lambda terms contain no observations; thus, it is a concept. It has the form of a complicated structural element whose application does not depend on its nodes' weights (Fehlmann, 2016, p. 326ff). For this characteristic, we call it *Lambda Concept*. For a proof of Barendregt's theorem for the graph model, see Fehlmann (Fehlmann, 1981).

Barendregt's Lambda calculus (3) means that programmable terms exist in the context of knowledge, making fixed rules part of general knowledge. For humans, this is nothing surprising; for machines, it is good to know that observation-independent rules exist like those used in social interactions between humans.

Similar to computer programming, you can write Lambda terms that contain no open variable terms, or you can write programs that process observations.

Fixpoint Combinators

Given any combinatory term Z, the *Fixpoint Combinator* $\mathbf{Y}$ generates a combinatory term $\mathbf{Y} \cdot Z$, called *Fixpoint of* Z, that fulfils $\mathbf{Y} \cdot Z = Z \cdot (\mathbf{Y} \cdot Z)$. This means that Z can be applied as many times as wanted to its fixpoint and still yields back the same combinatory term.

In linear algebra, such fixpoint combinators yield an eigenvector solution to some problem Z; for instance, when solving a linear matrix. It is therefore tempting to say, that $\mathbf{Y} \cdot Z$ is a solution for the problem Z.

Using Lambda Calculus notation, the fixpoint combinator can be written as:

$$\mathbf{Y} := \lambda \mathrm{f}. \left(\lambda \mathrm{x}.f \cdot (x \cdot x)\right) \cdot \left(\lambda \mathrm{x}.f \cdot (x \cdot x)\right) \tag{4}$$

Translating (4) into a combinator term in the graph model proves possible. It becomes a bit lengthy but demonstrates how Combinatory Logic works (Fehlmann & Kranich, 2022).

However, the fixpoint combinator is not the solution to all our problems. When applying $\mathbf{Y}$, or any other equivalent fixpoint combinator to a combinatory term Z, reducing the term by repeatedly using rule (2) does not always terminate. An infinite loop can occur, and sometimes must it occur, otherwise Turing would be wrong and all finite state machines would reach a finishing state (Turing, 1937).

Controlling Combinators

The concept of *Control* involves a *Controlling Operator* $\mathbf{C}$ which acts on a controlled object X by application $\mathbf{C} \cdot X$. Control means that the knowledge represented by X is completely known and described. It is a similar approach to establishing a fixpoint.

Accomplishing control can be formulated by (5):

$$\mathbf{C} \cdot X = X \tag{5}$$

The equation (5) is a theoretical statement, usually referring to an infinite loop process. For solving practical problems, X must be approximated by finite subterms.

Thus, the control problem is solved by a *Control Sequence* $X_0 \subseteq X_1 \subseteq X_2 \subseteq \ldots$, a series of finite subterms and the controlling operator $\mathbf{C}$, determined by (6):

$$X_{i+1} = \mathbf{C} \cdot X_i, \; i \in \mathbb{N} \tag{6}$$

starting with an initial X_0. This is called *Focusing*. The details can be found in Engeler (Engeler, 2019, p. 299). The controlling operator **C** gathers all faculties that may help in the solution. Like equation (4), controlling operators consist of structural content rather than of individual observations. The control problem is a repeated process of substitution, like finding the fixpoint of a combinator.

Within this framework, it is possible to define models for problem solving, software and system testing, and general reasoning (Engeler, 2019). Engeler constructs controlling combinators for a violinist, and for a mathematician. Both learn thanks to an *Attractor*. In the case of the violinist, the attractor points out when an action on the violin sounds right and when it is rather distorted. In the case of an artificial mathematician, advice is needed when the reasoning used is correct and logically based on axioms. Just guessing, or pretending without proof, is not acceptable, even if the claim is correct. This is supervised learning, not necessarily for a whole neural network, only for some part of it that is involved in the task.

Moreover, learning is something that happens internally in the brain. Theoretically, this could also happen in an artificial neural network, thanks to the Turing-completeness of the graph model. However, it is unclear how control combinators can become part of an artificial neural network. Structural combinators as in equation (4) are difficult to implement, because they would not appear in a weight matrix. Nor can we easily locate and identify Lambda concepts in a biological neural network. It touches on the question of where consciousness resides (Hepp, 2020).

Implementing Controlling Combinators

In theory, control combinators are easy to construct. Most controllers simply provide a feedback loop in which the appropriate parts of the neural network are trained to do the right thing. Engeler's violinist, for example, learns to play the violin by fine-tuning the parts of the neural network that control the actuators that hold the violin and pluck the string. It's easy to imagine a robot performing these tasks. All you have to do is give it feedback via a microphone and compare the result with a reference recording. Technically, the design and construction of such a violinist robot certainly is challenging.

All AI-enabled applications are a mixture of traditional programming, mostly in *Python*, and an AI-engine such as a neural network. Python is the most common language used for such tasks (Python, 2001-2024). According to the COSMIC standard (ISO/IEC 19761, 2019), these Python programs implement a few *Functional Processes* that move data groups between devices, permanent storage, and other applications. The data is processed, and new data groups are created. They furnish the data into the input layer of the AI-Engine and decode data from the output layer. ChatGPT, for example, is a loop with a prompt and a memory function that leads to a response. This implements a simple Lambda concept with only four data movements. Our violinist needs a few additional functional processes that connect and control sensors and actuators on the violin.

If the neural network is multi-purpose, then the violin skill uses only some parts of the graph representing the neural network. Panigrahi proposed recently how to identify areas in a neural network that contribute to some specific skills of this AI engine (Panigrahi et al., 2023). This is a necessary condition for deliberately defining control combinators in artificial neural networks because it must be able to identify those parts of the network that can be focused on an attractor. This is part of its reason for existence.

Controlling combinators can be thought of as Lambda concepts involving observations. The key to intelligence – in the sense of *νόος* – is that human beings use a lot of controlling combinators freely. Babys have natural controlling combinators that teach them how to communicate, eat, talk, and walk. Animals have similar, often more specific controlling combinators that enable them to survive without spending much time on learning. Why should this not become possible for artificial intelligence too?

The difficulty is that Lambda concepts are external to today's AI architecture. Artificial neural networks have a layered architecture with an input and an output layer. Between input and output layers, they consist of a sequence of attention and hidden layers holding the weights of the neural network nodes. The graph is ordered; it has no loop. Natural neural networks, in contrary, have loops. The graph model also has loops; otherwise, fixpoint and control operators would not exist.

Making AI Intelligent

The key to intelligent systems is to adapt architecture such that control combinators can be implemented. Such an architecture is most likely a combination of traditional functional processes with neural networks. Ideally, these functional processes are freely conceivable and can be configured on the spot, to meet changing requirements.

The programmer can prepare program patterns, implementing control combinators. This means its functional processes compare the output of the AI engine with some reference and prepare an input vector for the AI engine that reflects the learning. Any deep learning strategy can be used, whatever fits the problem domain best. While these control combinators can be programmed by AI engineers, the system should be able to select suitable control combinators of its own.

The Role of the Convergence Gap

Controlling combinators can be prepared to act on certain specific observations, identifying the relevant skills. The *Convergence Gap* is the numerical distance measure that specifies how good a skill had been learned. The problem is how to assess the responses of the AI engines with suitable references. This means setting the goal for the intelligent system what to learn.

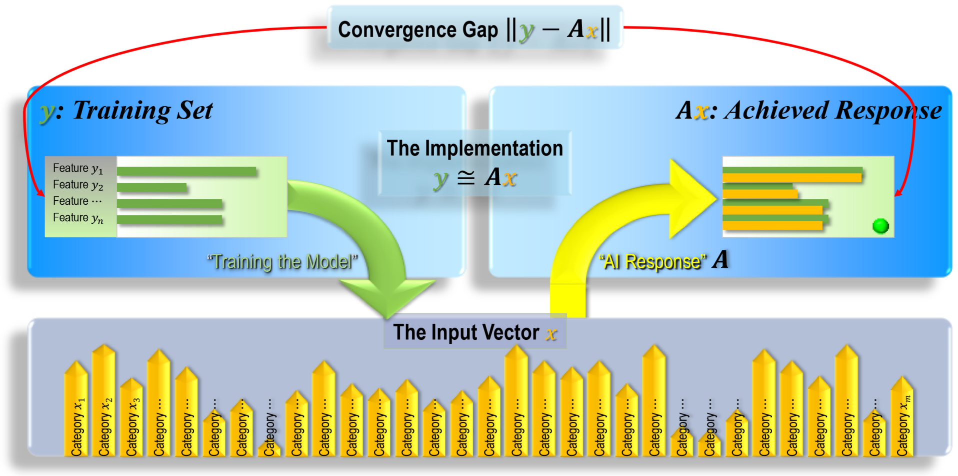

Figure 1: Deep learning for a skill with four features.

Figure 1 describes the principle of deep learning for an artificial neural network. From the training set, a functional process provides an input vector to the AI engine, and the output vector must be translated in terms of the required features.

The problem, as ever, is to define the goal. Sometimes, it is relatively easy, for instance when the goal is making customers happy. The needs of the customer can be assessed using standard methods from *Quality Function Deployment* (QFD) (Fehlmann & Mazur, 2016). For the violinist, and for most cyber-physical systems, getting it right is not so easily measurable. Sure, you need a reference, but when does the violin sound like its master reference? People have not a spectrometer built into their ears, they rather learn recognizing good musical sound from experience and exposure.

Therefore, it is better to start with easy cases and then increase the difficulty. For example, a robot learning to manipulate physical objects may have a convergence gap measured in mechanical precision. Or an autonomous car that learned driving in San Francisco can adapt to the way traffic flows in Split, Naples, Palermo, or even Delhi (Gent, 2024).

Empirical Methods for Programming Intelligent Systems

What otherwise can become utterly expensive – collecting and using training data for many individual settings – can therefore be left to an intelligent system that incorporates the necessary functional processes for training skills. These functional processes implement the controlling combinator needed for a specific skill.

The program scheme we use for the controlling combinator depends on four functional processes:

- The skills definition – i.e., how do we identify the skills that we want to improve

- The vector prompt – i.e., the vector furnished to the neural network to stimulate a response
- The convergence gap – i.e., the measurable variation between skill goals and skills achieved
- The nodes' weights adjustment – i.e., the deep learning strategy to improve the AI response

These functional processes depend on two parameters: the skill definition, which includes the convergence gap definition that must be provided externally. Then the intelligent system can do the fine-tuning autonomously by gaining control over certain capabilities. It needs feedback from attractors in the external world, and these attractors consist of the functional processes needed to compute the convergence gap and feed the result back to the AI engine for improvement.

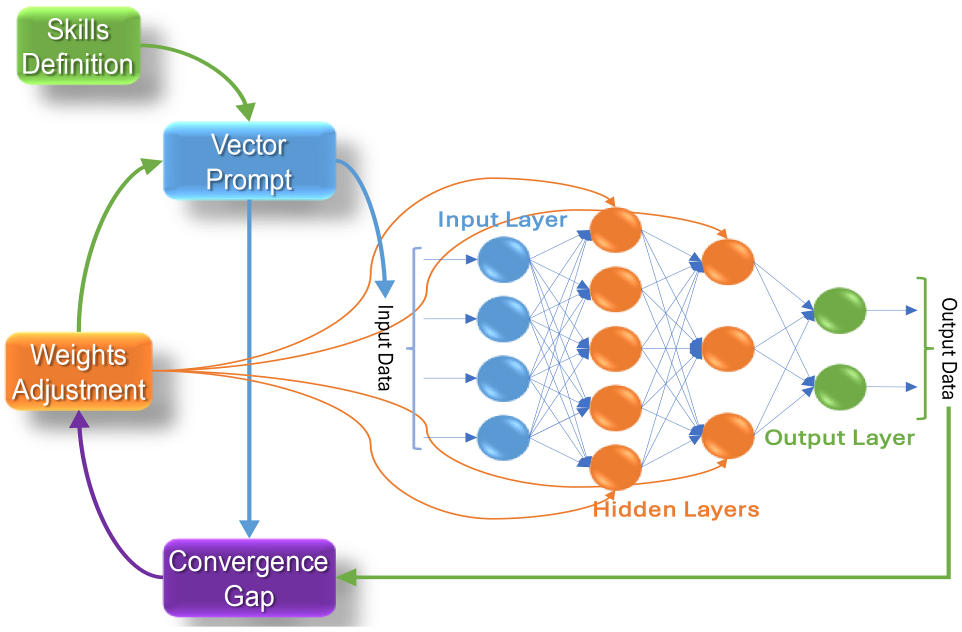

Figure 2: Empirical methods for programming intelligent systems.

Thus, intelligent systems are a joint venture between AI engines and traditional functional processes.

Future Tasks

Hybrid AI-enabled systems that implement concepts, i.e., traditional programs that operate artificial neural networks, are commonplace. Closing the loop within intelligent systems instead of traditional deep learning by AI engineers is the new challenge. Because of their layered architecture, artificial neural networks cannot implement control combinators, except loop-free ones. But loop-free is not interesting with respect to real intelligence in the sense of $\nu o \upsilon \sigma$. The ability to learn from feedback is a characteristic of life on Earth. It is possible, though not easy, to build intelligent systems with this capability.

We have prepared a number of sample cases for intelligent systems that can be freely used and inspected by interested parties. (Fehlmann, 2024).

CONCLUSION

What exactly does "intelligent" mean? We cannot be sure that we have met all the criteria that the gods, according to Homer, used to consider an android robot to be $\nu o \upsilon \sigma$. But we have added at least one sign of intelligence, namely the ability to learn adaptive behavior and new skills, to fine-tune the internal workings of the intelligent system, almost autonomously. To do this, we looked at relatively old concepts from theoretical computer science.

It may be that this is not only an advantage when trying to understand and use older concepts, even if they were not very successful in their time. The Perceptron of 1943 was a great vision, but far beyond the technical limitations of its time. The power of the graph model of combinatorial logic, on the other hand, is something to be explored further.

ACKNOWLEDGMENT

The authors would like to thank Nelio Fehlmann, born 30. March 2023, for showing us how a baby learns; to Lab42 in Davos for asking excellent questions and promoting the ARC Challenge (Chollet, 2019), and to the anonymous reviewers. Special thanks to Erwin Engeler for his suggestions and availability for very valuable discussions with his former student.

REFERENCES

Barendregt, H. & Barendsen, E., 2000. *Introduction to Lambda Calculus.* Nijmegen: University Nijmegen.

Bimbó, K., 2012. *Combinatory Logic - Pure, Applied and Typed.* Boca Raton, FL: CRC Press.

Chollet, F., 2019. *On the Measure of Intelligence,* Cornell University, Ithaca, NY: arXiv:1911.01547 [cs. AI].

Curry, H., Hindley, J. & Seldin, J., 1972. *Combinatory Logic, Vol. II.* Amsterdam: North-Holland.

Engeler, E., 1981. Algebras and Combinators. *Algebra Universalis,* Band 13, pp. 389–392.

Engeler, E., 2019. Neural algebra on "how does the brain think?". *Theoretical Computer Science,* Band 777, pp. 296–307.

Fehlmann, T. M., 1981. *Theorie und Anwendung der Kombinatorischen Logik,* Zürich, CH: ETH Dissertation 3140–01.

Fehlmann, T. M., 2016. *Managing Complexity – Uncover the Mysteries with Six Sigma Transfer Functions.* Berlin, Germany: Logos Press.

Fehlmann, T. M., 2020. *Autonomous Real-time Testing – Testing Artificial Intelligence and Other Complex Systems.* Berlin, Germany: Logos Press.

Fehlmann, T. M., 2024. *Intelligent Systems.* [Online] Available at: https://web.tresorit.com/l/AXX78#FaBkGqfY2cF_JsVmX70_ng

Fehlmann, T. M. & Kranich, E., 2020. Intuitionism and Computer Science – Why Computer Scientists do not Like the Axiom of Choice. *Athens Journal of Sciences,* 7(3), pp. 143–158.

Fehlmann, T. M. & Kranich, E., 2022. The Fixpoint Combinator in Combinatory Logic - A Step towards Autonomous Real-time Testing of Software?. *Athens Journal of Sciences,* 9(1), pp. 47–64.

Fehlmann, T. M. & Kranich, E., 2024. *The Neural Algebra and its Impact on Design and Test of Intelligent Systems.* Palermo, IHSI 2024 Open Access, AHFE International, USA.

Fehlmann, T. M. & Kranich, E., Preprint, 2024. A General Model for Representing Knowledge - Intelligent Systems Using Concepts. *ATINER Journal of Science.*

Fehlmann, T. M. & Mazur, G., 2016. *Using AHP in QFD - The Impact of the New ISO 16355 Standard.* Boise, ID, International Council for QFD (ICQFD).

Gao, Y. et al., 2024. *Retrieval-Augmented Generation for Large Language Models: A Survey,* Cornell University, Ithaca, NY: arXiv:2312.10997 [cs. CL].

Gent, E., 2024. Startups Say India Is Ideal for Testing Self-Driving Cars. *IEEE Spectrum*, 24 April.

Gödel, K., 1931. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik,* 38(1), pp. 173–198.

Hepp, K., 2020. Space, Time, Categories, Mechanics, and Consciousness: On Kant and Neuroscience. *Journal of Statistical Physics,* Band 180, pp. 896–909.

Hinton, G., 2012. A Practical Guide to Training Restricted Boltzmann Machines. In: G. Montavon & K. Müller, Hrsg. *Neural Networks: Tricks of the Trade.* Berlin, Heidelberg: Springer Lecture Notes in Computer Science, vol. 7700, pp. 599–619.

Homer, 1898. *The Iliad.* London, New York, and Bombay: Longmans, Green and Co.

ISO/IEC 19761, 2019. *Software engineering - COSMIC: a functional size measurement method,* Geneva, Switzerland: ISO/IEC JTC 1/SC 7.

Langville, A. N. & Meyer., C. D., 2006. *Google's PageRank and Beyond: The Science of Search Engine Rankings.* Princeton, NJ: Princeton University Press.

Minsky, M. & Papert, S., 1972. *Perceptrons: An Introduction to Computational Geometry.* 2nd edition with corrections Hrsg. Cambridge (MA): The MIT Press.

Panigrahi, A., Saunshi, N., Zhao, H. & Arora, S., 2023. *Task-Specific Skill Localization in Fine-tuned Language Models,* Cornell University, Ithaca, NY: arXiv:2302.06600v2 [cs. CL].

Python, S. F., 2001–2024. *Python Setup and Usage.* [Online] Available at: https://docs.python.org/3/using/index.html

Turing, A., 1937. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society,* 42(2), pp. 230–265.

Winograd, T., 1972. *Understanding Natural Language.* New York: Academic Press.

Wolfram, S., 2023. *What is ChatGPT doing... and Why Does it Work?.* Champaign, IL: Wolfram Media, Inc.

Zachos, E., 1978. *Kombinatorische Logik und S-Terme,* Zurich: ETH Dissertation 6214.

Zhong, V. et al., 2022. *Improving Policy Learning via Language Dynamics Distillation,* Cornell University: arXiv:2210.00066v1.