Combinatory Logic as a Model for Intelligent Systems Based on Explainable AI

Thomas Fehlmann
Euro Project Office AG
8032 Zurich, Switzerland
E-mail: thomas.fehlmann@e-p-o.com

Eberhard Kranich
Euro Project Office
47051 Duisburg, Germany
E-mail: eberhard.kranich@t-online.de

***Abstract*—Directed graphs such as neural networks can be described by *Arrow Terms* that link a finite set of incoming nodes to some response node. Scott and Engeler have shown that its powerset is a model for *Combinatory Logic*. This algebra is called *Graph Model of Combinatory Logic*. Since Combinatory Logic is Turing-complete, the model explains both traditional programming logic as well as neural networks such as the brain or *Artificial Neural Networks* as used in a *Large Language Model*. The underlying graph model is a general model for all kinds of knowledge. The graph model would yield a powerful AI-tool if used as a blueprint for implementing AI. *Chain of Thoughts* would come for free, and explainability with it. However, its performance would make such a tool impractical and useless. The paper proposes a combined approach for adding explainability to AI and creating *Intelligent Systems*. It is the strategy humans use when they try to explain their ideas. First, the generative power of neural networks is used to produce an idea or solution. Next, humans create a chain of thoughts that explain such ideas to others and try to provide evidence. AI could follow the same strategy. The architecture of such intelligent systems consists of two distinct elements: a well-trained artificial neural network for observing and generating solution approaches, and a controlling engine for fact checking and reliability assessment.**

***Keywords*—Intelligent Systems; Chain-of-Thought (CoT); Explainable AI (XAI); Artificial Neural Networks (ANN); Deep Neural Network (DNN); Combinatory Logic; Quality Function Deployment (QFD).**

I. INTRODUCTION

This paper is a revised version of the author's contribution to the 1st International Conference on Systems Explainability, held in Valencia, Spain, in autumn 2024 [1].

A. Short History of AI and its Philosophical Background

In the early 20th century, there were some shocking events taking place in mathematical logic and natural science. Gödel [2], when trying to solve some of Hilbert's 23 problems, detected that predicate logic, something with a long history dating back to the ancient Greeks, is undecidable. This insight gave birth to theoretical computer science, including the theory of computation, founded by Turing [3]. For a modern compilation, see Raatikainen [4].

Schönfinkel and Curry [5] developed *Combinatory Logic* to avoid the problems introduced when using logical quantifiers, and Church invented *Lambda Calculus* as a rival formalism [6]. Scott and Engeler developed the *Graph Model* [7], based on *Arrow Terms*, and proved that this is a model of combinatory logic. This means that you can combine sets of arrow terms to get new arrow terms, and that combinators, accelerators, and constructors can be used to create new elements of algebra.

Graphs in the form of neural networks appeared already at the origins of *Artificial Intelligence* (AI). Its first instantiation in modern times was the *Perceptron*, a network of neurons postulated by Rosenblatt [8]. It later became a directed graph [9]. Rosenblatt was also the first who postulated concepts, among perception and recognition, as constituent parts of AI [8, p. 1].

Since its origins, AI has experienced difficulties; however, today there are many AI applications that provide value for the user. In some areas, training an AI model is simpler and more rewarding than finding and programming an algorithm.

For instance, AI-powered visual recognition systems excel in recognizing and classifying objects, following the ideas established by Rosenblatt [8]. However, they have difficulty recognizing temporal dependencies and are unable to combine what they have learned, although attempts have been made to develop methods using sequential data and the ability to capture temporal patterns. AI lacks what humans use in such cases: a concept.

Logical skills such as inference and deduction provide quite a challenge, as exemplified by the ARC Price challenge, a sort of intelligence test for AI models, proposed by Chollet [10]. A *Large Language Model* (LLM) easily summarizes texts or books but it still does not understand what is written in it, in the sense that the US National Council of English Teachers calls *Literacy*, see [11], [12].

Artificial Neural Networks (ANN) can be divided into four types: Recurrent Neural Network (RNN), Fuzzy Neural Network (FNN), Convoluted Neural Network (CNN) and Deep Neural Network (DNN). DNN are the most successfully used for LLM and thus the most important type of ANN regarding explainability because they rely on many hidden layers. Among the rapidly developing literature, Gerven & Bothe's classification are a good start [13]. *Natural Neural Networks*, in analogy to ANNs, are abbreviated by NNN.

IBM defines *Explainable Artificial Intelligence* (XAI) as a set of processes and methods that allow human users to comprehend and trust the results created by machine learning

algorithms [14]. Current approaches to XAI attempt to use statistical correlations as a basis for reasoning. From a theoretical perspective, this is unlikely to work, because of Gödel [2]. However, engineers try to use statistical methods to circumvent Gödel's undecidability. Sometimes the results look convincing. Dallanoce [15] compiled a list of available processes and methods for XAI.

Artificial General Intelligence (AGI) is a type of artificial intelligence (AI) that falls within the lower and upper limits of human cognitive capabilities across a wide range of cognitive tasks. The creation of AGI is a primary goal of AI research and companies such as OpenAI and Meta, but what exactly AGI refers to is controversial [16].

Current attempts towards AGI focus on the investigation of *Chain of Thoughts* (CoT). Using an LLM based on DeepSeek with CoT enabled yields the feeling that the LLM does some "reasoning" because it displays the intermediate results obtained with the processing of some query [17]. The real innovation behind DeepSeek is using a hash function to avoid processing useless branches in an LLM. This is not what reasoning really is, namely the use of logic based on factual knowledge to find previously unknown answers. The hash function is still based on statistics from a suitable training set [18].

B. Research Questions

The aim of this paper is to recall prior work in logic and AI to understand how neural networks work. To do this, we investigate the following three research questions:

RQ 1: How are neural networks and especially DNNs linked to the graph model?
RQ 2: Does CoT relate to a sequence of arrow terms?
RQ 3: Can the graph model explain AI?

The motivation for this is that we are experiencing the fourth AI hype in sixty years and that its acceptance in society is currently transitioning from admiration to rejection. Because the nature of AI is poorly understood not only by society but also by the AI research community. We believe that the graph model is an excellent way to understand what intelligence is, both natural and artificial. However, it is not an answer to how to construct XAI.

C. Paper Structure

We first explain combinatory logic (Section II) and the motivation for building a model (Section III). Then we compare DNNs with graphs and explain how arrow schemes represent what a DNN does and have an outlook on the architecture of intelligent systems (Section IV). Finally, we present the method for designing intelligent systems (Section V) and explain how we want to go ahead (Section VI).

II. COMBINATORY LOGIC

In the past decades, there has been a lack of attention and consequently of publications on *Combinatory Logic.* Nevertheless, it explains quite a bit what artificial intelligence can do and what not.

A. Combinatory Logic and Axiom of Choice

Combinatorial Logic is a notation that eliminates the need for quantified variables in mathematical logic, and thus the need to explain what the meaning of existential quantifiers $\exists x \in M$ is, see Curry [5] and [19]. Eliminating quantifiers is an elegant way to avoid the *Axiom of Choice* [20] in its traditional form. Combinatory Logic can be used as a theoretical model for computation and as design for functional languages (Engeler [21]); however, the original motivation for combinatory logic was to better understand the role of quantifiers in mathematical logic.

Combinatory logic is based on *Combinators* which were introduced by Schönfinkel in 1920. A combinator is a higher-order function that uses only functional applications, and earlier defined combinators, to define a result from its arguments.

The combination operation is denoted as $M \bullet N$ for all combinatory terms M, N. To make sure there are at least two combinatory terms, we postulate the existence of two special combinators **S** and **K**.

They are characterized by the following two properties (1) and (2):

$$\mathbf{K} \bullet P \bullet Q = P \tag{1}$$

$$\mathbf{S} \bullet P \bullet Q \bullet R = P \bullet Q \bullet (P \bullet R) \tag{2}$$

P, Q, R are terms in combinatory logic. The combinator **K** acts as projection, and **S** is a substitution operator for combinatory terms. Equations (1) and (2) act like axioms in traditional mathematical logic.

Like an assembly language for computers, or a Turing machine, the **S**-**K** terms become quite lengthy and are barely readable by humans, but they work fine as a foundation for computer science. The power of these two operators is best understood when we use them to define other, handier, and more understandable combinators.

The identity combinator for instance is defined as

$$\mathbf{I} := \mathbf{S} \bullet \mathbf{K} \bullet \mathbf{K} \tag{3}$$

Indeed, $\mathbf{I} \bullet M = \mathbf{S} \bullet \mathbf{K} \bullet \mathbf{K} \bullet M = \mathbf{K} \bullet M \bullet (\mathbf{K} \bullet M) = M$. Association is to the left. Moreover, **S** and **K** are sufficient to build a Turing machine. Thus, combinatory logic is Turing-complete. For proof, consult Barendregt [22, pp. 17-22].

B. Functionality by the Lambda Combinator

Curry's *Lambda Calculus* [23] is a formal language that can be understood as a prototype programming language. The **S**-**K** terms implement the lambda calculus by recursively defining the *Lambda Combinator* $\mathbf{L_x}$ for a variable x as follows:

$$\begin{gathered} \mathbf{L_x} \bullet x = \mathbf{I} \\ \mathbf{L_x} \bullet Y = \mathbf{K} \bullet Y \text{ if } Y \text{ different from } x \\ \mathbf{L_x} \bullet M \bullet N = \mathbf{S} \bullet \mathbf{L_x} \bullet M \bullet \mathbf{L_x} \bullet N \end{gathered} \tag{4}$$

The definition holds for any term **x** of combinatory logic. Usually, on writes suggestively $\lambda x.M$ instead of $\mathbf{L_x} \bullet M$, for any combinatory term M. *Lambda Terms* $\lambda x.M$ offer the

possibility of programmatic parametrization. Note that $\lambda x.M$ is a combinatory term, as proofed by (4), and that this introduces a kind of variable in combinatory logic with precisely defined binding behavior.

The Lambda combinator allows writing programs in combinatory logic using a higher-level language. When a lambda term is compiled, the resulting combinatorial term looks like machine code in traditional programming languages.

C. The Fixpoint Combinator

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

In linear algebra, such fixpoint combinators yield an eigenvector solution $\mathbf{Y} \bullet Z$ to some problem Z.

According to Barendregt in his textbook about Lambda calculus [22, p. 12], the fixpoint combinator can be written as

$$\mathbf{Y} := \lambda f.\left(\lambda x.f \bullet (x \bullet x)\right) \bullet \left(\lambda x.f \bullet (x \bullet x)\right) \qquad (5)$$

Translating (5) into an $\mathbf{S}$– $\mathbf{K}$ term demonstrates how combinatory logic works, see [24].

When translated into arrow terms, the fixpoint combinator contains loops. Fixpoint operations are related to infinite loops, thus, to programming constructions that never end and have no normal form. Applying $\mathbf{Y}$, or any equivalent fixpoint combinator to a combinatory term Z, usually does not terminate. An infinite loop can occur, and must sometimes occur, otherwise Turing would be wrong and all finite state machines would reach a finishing state [3].

D. A few More Sample Combinators

The following samples are taken from Zachos 1978 [25], where all proofs are given:

- Composition:
  $\mathbf{B} \bullet P \bullet Q \bullet R = P \bullet Q \bullet R$ by
  $\mathbf{B} := \mathbf{S} \bullet (\mathbf{K} \bullet \mathbf{S}) \bullet \mathbf{K}$
- Exchange of arguments:
  $\mathbf{C} \bullet P \bullet Q \bullet R = P \bullet R \bullet Q$ by
  $\mathbf{C} := \mathbf{S} \bullet (\mathbf{B} \bullet \mathbf{B} \bullet \mathbf{S}) \bullet (\mathbf{K} \bullet \mathbf{K})$
- Argument identification:
  $\mathbf{W} \bullet P \bullet Q = P \bullet Q \bullet Q$ by
  $\mathbf{W} := \mathbf{S} \bullet \mathbf{S} \bullet (\mathbf{K} \bullet \mathbf{I})$
- Composition:
  $\mathbf{\Phi} \bullet O \bullet P \bullet Q \bullet R = O \bullet (P \bullet R) \bullet (Q \bullet R)$ by
  $\mathbf{\Phi} := \mathbf{B} \bullet (\mathbf{B} \bullet \mathbf{S}) \bullet \mathbf{B})$
- Composition:
  $\mathbf{\Psi} \bullet O \bullet P \bullet Q \bullet R = O \bullet (P \bullet Q) \bullet (P \bullet R)$ by
  $\mathbf{\Psi} := \mathbf{B} \bullet (\mathbf{B} \bullet \mathbf{W} \bullet (\mathbf{B} \bullet \mathbf{C})) \bullet \mathbf{B} \bullet \mathbf{B} \bullet (\mathbf{B} \bullet \mathbf{B})$
- Fixpoint Combinator:
  $\mathbf{Y} \bullet R = R \bullet (\mathbf{Y} \bullet R)$ by
  $\mathbf{Y} := \mathbf{W} \bullet \mathbf{S} \bullet (\mathbf{B} \bullet \mathbf{W} \bullet \mathbf{B})$

There is no negation combinator, because with a negation $\boldsymbol{N}$ we would have $\mathbf{Y} \bullet \boldsymbol{N} = \boldsymbol{N} \bullet (\boldsymbol{Y} \bullet \boldsymbol{N})$. This contra-intuitive example explains why so few people dare to work with combinatory logic. However, it also strengthens our point that it is highly rewarding to try it.

It is a specific human behavior to identify complicated behavior with simple explanations, such as "Exchange of arguments." If you expand that combinator, it would be near to unreadable; same with the fixpoint operator $\mathbf{Y}$, as shown in [24].

III. The Graph Model of Combinatory Logic

The graph model is a versatile model for knowledge in all its different forms. It is highly recursive and Turing-complete, which means it can also be used to describe conventional algorithmic programming. The LISP language was once created to allow programming in a framework close to the graph model [26].

A. A Logic Needs a Model

A *Model* for a logical structure is a set-theoretic construction that has the properties postulated for the logic and can be proved to be non-empty. Then it means that such logic makes sense as far as it describes an existing structure and can be used to prove something about the model.

Let $\mathcal{L}$ be a non-empty set. Engeler [7] defined a *Graph* as the set of ordered pairs:

$$\langle \{a_1, a_2, \dots, a_m\}, b \rangle \qquad (6)$$

with $a_1, a_2, \dots, a_m, b \in \mathcal{L}$. We write $\{a_1, \dots, a_m\} \to b$ for the ordered pair to make notation mnemonic, i.e., referring to directed graphs, and call them *Arrow Terms*. These terms describe the constituent elements of directed graphs with multiple origins and a single node. We refer to $\mathcal{L}$ as *Observations,* and to terms $\{a_1, \dots, a_m\} \to b$ as *Concepts,* i.e., a non-empty finite set of arrow terms with level 1 or higher.

We extend the definition of arrow terms to a powerset by including all formal set-theoretic objects recursively defined as follows:

$$\begin{gathered}\text{Every element of } \mathcal{L} \text{ is an arrow term.}\\ \text{Let } a_1, \dots, a_m, b \text{ be arrow terms.}\\ \text{Then } \{a_1, \dots, a_m\} \to b \text{ is also an arrow term.}\end{gathered} \qquad (7)$$

The left-hand side of an arrow term is a finite set of arrow terms, and the right-hand side is a single arrow term. This definition is recursive. Elements of $\mathcal{L}$ are also arrow terms. The arrow, where present, should suggest the ordering in a graph, not logical imply.

B. Einstein-Notation for Arrow Terms

To avoid the many set-theoretical parenthesis, the following notation, called *Arrow Schemes*, is applied, in analogy to the Einstein notation [27, p. 6]:

- a_i for a finite set of arrow terms, i denoting some Choice Function selecting finitely many specific terms out of a set of arrow terms a.
- a_1 for a singleton set of arrow terms; i.e., $a_1 = \{a\}$ where a is an arrow term. (8)
- $\emptyset$ for the empty set, such as in the arrow term $\emptyset \to a$.
- $a_i + b_j$ for the union of two observation sets a_i ,b_j.

The application rule for M and N now reads:

$$M \bullet N = (a_i \to b) \bullet N = \{b | \exists a_i \to b \in M;\ a_i \subset N\} \qquad (9)$$

Arrow schemes always represent sets of arrow terms. $(a_i \to b) \subset M$ is the subset of level 1 arrow terms in M, provided $a_i \in M$ and $b \in M$. Thus, $(a_i \to b)_j$ denotes a concept, together with two choice functions i, j. Each set element has at least one arrow.

The choice function i chooses specific observations a_i out of a (larger) set of observations a. This is what Zhong describes as *Grounding* when linking observations to real-world objects [28]. In AI, grounding is crucial for linking AI engines to the real world. If a denotes knowledge, i.e., an infinite set of arrow terms of any level, a_i can become part of a concept consisting of specific arrow terms referring to some specific object, specified by the choice function i. Choice functions therefore have the power of focusing knowledge on specific objects in specific areas. That makes choice functions interesting for intelligent systems and AI.

There is a conjunction of choice functions, thus $a_{i,j}$ denotes the union of a finite number of grounded arrow schemes:

$$a_{i,j} = a_{i,1} \cup a_{i,2} \cup \ldots \cup a_{i,m} = \bigcup_{k=1}^{m} a_{i,k} \qquad (10)$$

There is also cascading of choice functions. Assume $N = (a_j \to b)_k$, then:

$$M = \left(\left((a_j \to b)_k \to b_i\right)_l \to c\right) \text{ and } M \bullet N = (b_{i_l} \to c) \qquad (11)$$

The choice function might be used for grounding an arrow scheme to observations.

An arrow scheme without outer indices represents a potentially infinite set of arrow terms. Thus, writing a, we mean knowledge about an observed object. Adding an index, a_j, indicates such a grounded object together with a choice function j that chooses finitely many specific observations or knowledge.

While on the first glimpse, the Einstein notation seems like just another way of denoting arrow terms, for representing such data in computers it means that the simple enumeration of finite data sets is replaced by an intelligent choice function providing grounding that must be computed and can be either programmed or guessed by an intelligent system.

For practical applications, the choice function is an important part of deep learning. It means learning by generalization. The more choices you get on the left-hand side, the more knowledge you acquire. The ARC price competition for instance is easily solvable if we can generalize our choice functions good enough, drawing conclusions from the samples into general rules. However, generalization is not easily available with current AI technology. *Controlling Combinators*, see Section IV.C, are a workaround.

C. The Graph Model of Combinatory Logic

The algebra of observations represented as arrow terms is a combinatory algebra and thus a model of combinatory logic. The following definitions demonstrate how the graph model implements Curry's combinators **S** and **K** fulfilling equations (1) and (2), following [5].

- $\mathbf{I} = a_1 \to a$ is the Identification, i.e., $(a_1 \to a) \bullet b = b$
- $\mathbf{K} = a_1 \to \emptyset \to a$ selects the 1st argument:
  $\mathbf{K} \bullet b \bullet \mathrm{c} = (b_1 \to \emptyset \to b) \bullet b \bullet \mathrm{c} = (\emptyset \to b) \bullet \mathrm{c} = \mathrm{b}$
- $\mathbf{KI} = \emptyset \to a_1 \to a$ selects the 2nd argument:
  $\mathbf{KI} \bullet b \bullet \mathrm{c} = (\emptyset \to c_1 \to c) \bullet b \bullet \mathrm{c} = (c_1 \to c) \bullet c = \mathrm{c}$
- $\mathbf{S} = \left(a_i \to \left(b_j \to c\right)\right)_1 \to (d_k \to b)_i \to \left(a_i + b_{j,i} \to c\right)$

(12)

Therefore, the algebra of observations is a model of combinatory logic. The interested reader can find complete proofs in Engeler [7, p. 389].

The *Lambda Theorem* from Barendregt [23] says that with **S** and **K,** an abstraction operator can be constructed that adds algorithmic skills to knowledge represented as arrow schemes, following equation (4).

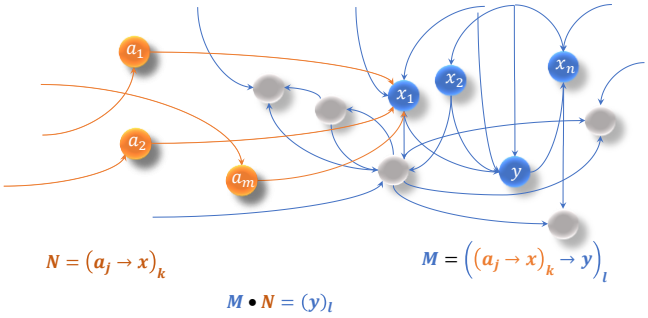

Figure 1. A Neural Network becomes a Combinatorial Algebra

As the name "graph model" suggests, arrow terms are an algebraic way of describing neural networks. Thus, something that nature uses to acquire and work with knowledge.

Figure 1 illustrates the effect of the combination according to equation (9). It becomes apparent that the graph model describes graphs indeed, with loops. Repeatedly applying equation (9) leads to what we perceive as the "response of a neural network". The combination of knowledge and combinators thus plays a significant role in AI.

However, Figure 1 is not only a picture of an abstract graph. It can also be understood as a part of a *Deep Neural Network* (DNN) – or of a *Natural Neural Network* (NNN). Engeler [29] associated neuroscience with the graph model in 2019, by explaining how a brain works. He used the graph model as an algebraic representation of NNN.

IV. TOWARDS INTELLIGENT SYSTEMS

Barceló et al. has shown in 2019 that modern neural network architectures are Turing-complete [30]. This is also a property of the graph model but not of every DNN. An architecture for intelligent systems should be suitable for using conventional algorithmic programming instead of complex arrow notations.

A. Solving the World Formula

Artificial neural networks learn continuously by using corrective feedback loops to improve their predictive analytics. Neural networks perform supervised learning tasks, building knowledge from training data where the right answer is provided in advance. In contrast, in unsupervised learning, algorithms learn patterns exclusively from unlabeled data [31]. There exist mixed forms; a famous example of semi-supervised learning has led to the creation of ChatGPT [32].

In both cases, the principle is the same as with *Six Sigma Transfer Functions* (SSTF) [33]: One has to solve an equation (13), where the expected response $\boldsymbol{y}$ is known but neither the required controls $\boldsymbol{x}$ nor the transfer function $\boldsymbol{A}$ itself, which cause this response, are known. Transfer functions are abundant in technology and science – just to mention the *Fast Fourier Transform* (FFT) of audio and video signals from analog to digital [34] – and AI-enabled applications belong also to that category. In either case, the problem to be solved is:

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \tag{13}$$

The equation (13) is often called the "World Formula" [35]. In the case of AI, the world formula describes *Deep Learning* (DL), i.e., the process of parametrizing the model so that it provides the expected answers.

In AI, the transfer function $\boldsymbol{A}$ is usually represented as a large sparse matrix. For small dimensions, the easiest way to solve equation (13) is the Eigenvector method used by Saaty for the *Analytic Hierarchy Process* (AHP), for decision making method [36]. The method also works for *Quality Function Deployment* (QFD) [33, p. 34].

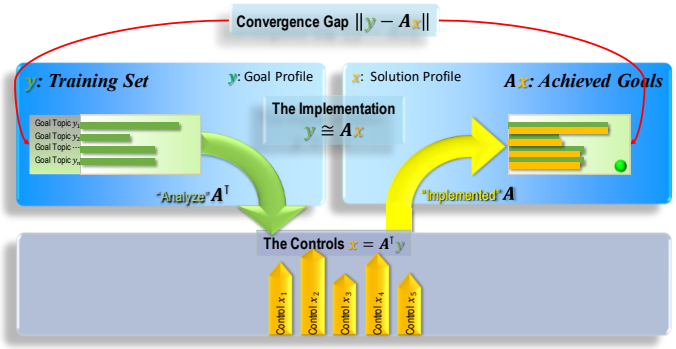

Figure 2. Solving the World Formula

The idea of the Eigenvector solution method is to calculate the *Principal Eigenvector* $\boldsymbol{y_E}$ with the property:

$$\boldsymbol{y_E} = \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y_E} \tag{14}$$

The principal eigenvector exists due to the *Perron-Frobenius* theorem [33, p. 365]. Setting $\boldsymbol{x_E} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{y_E}$ yields an approximate solution to equation (13), using equation (14), provided that $\boldsymbol{y} \cong \boldsymbol{A}\boldsymbol{x_E}$ is close enough. The Euclidean distance (15) is called the *Convergence Gap*:

$$\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x_E}\| \tag{15}$$

The eigenvector method is not applicable for large AI matrices, because the solution is numerical and not algebraic. New methods suitable for large sparse matrices representing neural networks had to be invented, see for example Hinton [31].

The breakthrough for solving such matrices, and thus enabling machines for deep learning, happened in 2012 and involves breaking up those matrices into smaller pieces that can be managed in parallel. Its impact on humanity and society might become comparable with the FFT transform of 1977 that made the analog/digital conversion of audio and video in real-time possible and thus stood at the origins of the Internet of today.

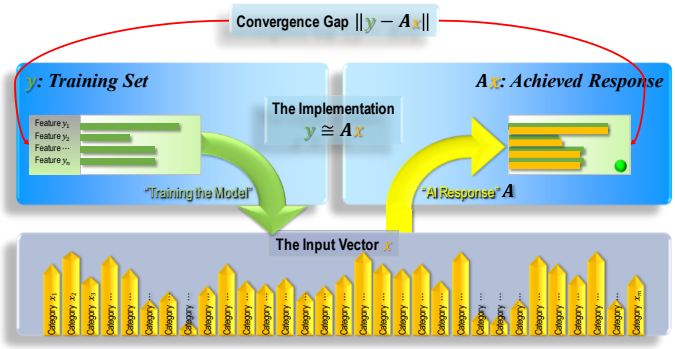

Figure 3. Deep Learning as a Transfer Function

Once the neural net has been sufficiently trained, the model can be used to predict responses not just for the training set but for any query submitted to the AI. The convergence gap in Figure 3, the vector distance between the true response and the response received, is what we consider the inaccuracy, or uncertainty, of the trained model.

B. How Arrow Schemes describe DNNs

While it is obvious how an NNN is represented by arrow schemes, this is not equally clear for ANNs. The reason is that directed graphs contain loops while looping in ANNs is very restricted. There exist certain architectures for ANNs that allow for loops, within narrow limits; however, a *Multi-Layered Perceptron* (MLP) as used for LLMs does not [13].

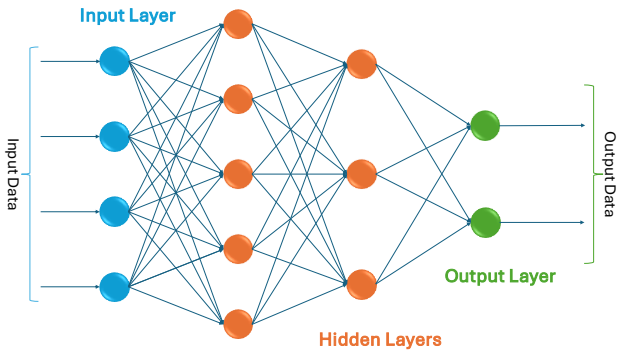

Figure 4. Multi-Layered Perceptron as an DNN

Consequently, a DNN has only a limited ability to emulate an NNN. In principle, every arrow scheme $a_i \to b$ describes one node in a directed but not loop-free graph. Some arrow schemes describe algorithmic concepts such as in equation (12) or as explained in equation (5). Other arrow schemes simply connect observations a_i to some response b. General knowledge has many facets.

It would be wonderful if we had the ability to look at an LLM and identify arrow schemes for each node. This would add full explainability to AI, but unfortunately, this has no practical value. Neither combinatory terms nor arrow schemes have normal forms. Very often there is a wide variety of solutions that are equivalent but widely different; not only formally but also in effectiveness.

This makes explainability of AI difficult. The lack of normal forms blocks all attempts to find the one sequence of arrow schemes that explains what AI is doing. AI engineers have no other choice than trying to train their DNNs such that the response meets expectations but without exactly knowing what happens. It is comforting, however, that they share the same sad fate with neuroscientists. It is astonishing how long-forgotten theoretical results such as the lack of a normal form in combinatory logic yields economically relevant results, nowadays, in the evolving AI ecosystem. Consult Lachowski [37] for a survey of the performance challenges that occur around combinatory logic.

However, there is a famous saying that nothing is too difficult for the engineer ("Inventor of Anything"). Recent findings suggest that AI is capable of recognizing chains of thought that lead to the observation of a specific response [38]. This complements earlier findings that describe CoT as a prompting technique [17]. Thus, there exist AI architectures that allow us to identify at least some arrow schemes that describe what AI does. It is not necessarily the whole truth, just as it is not when people explain their thoughts to colleagues. But it should be enough to convince them.

Having a complete sequence of arrow schemes describing approximatively some DNN would lead to explainable AI that even is able to get certified for safety-critical applications. However, the problem with hidden layers remains. While the QFD method uses identifiable topics for each layer [39], an DNN has none; they remain hidden and unknown. Thus, much of the intermediate reasoning also remains hidden. RQ 2 remains at least partially unanswered. If the input data and response can only be captured by arrow schemes, the intermediate steps must be guessed based on domain knowledge, but it is not known exactly what the AI engine did consider. AI might change behavior and create hazardous changes to the hidden layers. Low-rank adaptation (LoRA) of LLMs is an attempt to limit such change [40]. In QFD, on the contrary, intermediate stages are identifiable based on their topic; for example, when deploying customer needs, we first go to user stories and then to testable features.

Another approach to better explainable AI is already well established: *Retrieval-Augmented Generation* (RAG) might avoid hallucinations for LLMs [41] by referencing knowledge databases and including them into the generation of responses. RAG impacts the architecture of intelligent systems by connecting neural networks to knowledge databases [42]. RAG corresponds to grounding arrow schemes using the choice function; RAG is indispensable for explainable AI.

This is the motivation for looking at AI architecture. In some way, it must be complemented by functionality that controls the behavior of AI. With such controls an AI-engine can perform safety-critical tasks. When certifying an AI-engine for safety, it is not necessary to convert all nodes of an DNN into arrow schemes, but we can focus on the overall result, because these results are not presented plainly but reviewed by a controlling combinator first. If an AI fails on such tasks, we do not have a white-box trace of all nodes including their arrow schemes that have contributed to this failure, but we are at least as good as with traditional safety-preserving methods and techniques.

C. The Architecture of Intelligent Systems

Intelligent systems using AI are based upon *Controlling Combinators*. Controlling combinators are derived from the idea behind fixpoint combinators, see equation (5) but refer to effective factual knowledge or to skills. Examples from Engeler include controlling combinators for learning mathematics, or for playing violin [29].

A *Controlling Operator* $\mathbf{C}$ acts on a controlled object X by its application $\mathbf{C} \bullet X$. Control means that knowledge needed to execute a task that is represented by arrow schemes in X is sufficiently well-known and described. This implies the need for a metric that measures the convergence gap. Note that $\mathbf{C}$ itself a term of the graph model of combinatory logic and thus a combinatory algebra term. Then, accomplishing control can be formulated by (16):

$$\mathbf{C} \bullet X = X \tag{16}$$

The equation (16) is a theoretical statement, referring to a potentially infinite loop. For solving practical problems, X must be approximated by finite subterms.

Thus, the control problem is solved by a *Control Sequence* $X_0 \subseteq X_1 \subseteq X_2 \subseteq \cdots$, a series of finite subterms and the controlling operator $\mathbf{C}$, starting with an initial X_0 and determined by (17):

$$X_{i+1} = \mathbf{C} \bullet X_i, i \in \mathbb{N} \tag{17}$$

This is called *Focusing*. The details can be found in Engeler [29, p. 299]. The controlling operator $\mathbf{C}$ gathers all faculties that may help in the solution. The inclusion operator in equation (17) is explained by the graph model. The control problem is a repeated process involving substitution, like finding the fixpoint of a combinator, and thus increasing the number of arrow schemes, and especially of choice functions, in the resulting focusing process.

Controlling combinators both collect and use empirical data for continuous training. Such an intelligent system incorporates the necessary functional processes for fine-tuning based on feedback received. For further details, please refer to the authors' paper on solving the control problem [43].

If the "Skills Definition" in Figure 5 is a training set, the program scheme represents *Deep Learning* [44]. In case the definition is linked to some feedback or hash function as in DeepSeek, it is *Reinforcement Learning* [45]. In all these

cases, the controlling combinators use the convergence gap; the measurable variation between actual behavior and expectations and requirements. In AI, the convergence gap is the same as in equation (15), but it is called "Loss Function". This term originates from *Signal Theory* and originally describes the loss of fidelity in analog sound transmission. Since the discovery of the *Fast-Fourier Transform* (FFT) [34], one understands that A/D-convergence is not a loss, but an acquisition of enough knowledge to reach a specific threshold for high-fidelity rendering of music. Deep learning uses the same principles.

Both come as (large) vectors and thus the Euclidean distance is easily computable. If learning is continuous, e.g., by experiences, by external feedback from a tutor, or by physical sensors, it is called an *Intelligent System*. Expectations and correct answers might also come from an external knowledge database, allowing the intelligent system to learn autonomously.

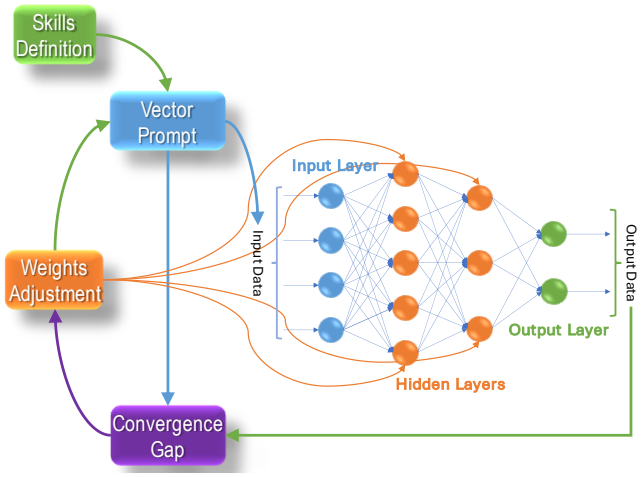

Figure 5. Controlling Combinators for self-learning intelligent systems

The architecture for RAG now extends. Instead of embedding the reference into response generation [42], and hoping it works, we set up functional processes for comparing LLM results with evidence from the knowledge database and calculating the convergence gap.

The convergence gap of such a system fully explains its behavior. Under well-defined conditions, such a system can be certified, even for safety critical tasks.

It is also possible to add more than one AI engine to an intelligent system, compare results and go forward with the most reliable one. Insufficient training, biases, and hallucinations therefore would become detectable.

Figure 6 shows an example of an intelligent system design that relies on two separate visual recognition engines analyzing the same scenario, one through a camera and the other through a Lidar. Such architecture requires that the reliability of each AI engine be known, under certain conditions, such as weather. In this way, the intelligent system can explain why it selected one or the other response.

If both AI engines provide an identical answer, this increases the overall reliability of the intelligent system's response significantly.

The graph model delivers the metrics for defining controlling combinators by inclusion, and it also allows us to combine knowledge and thus reliability correctly, by equation (9). This is discussed in the following section.

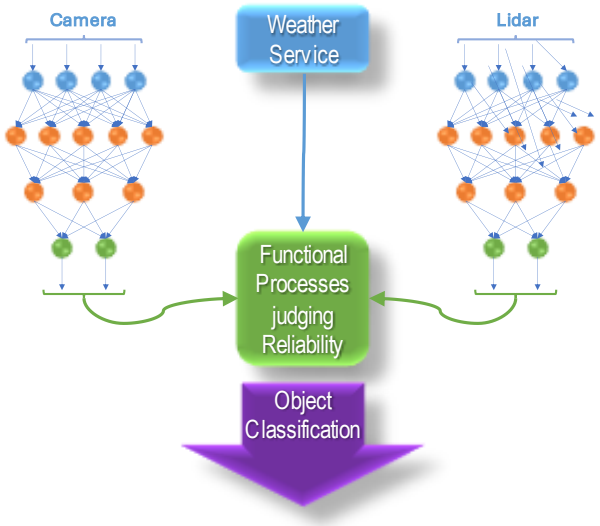

Figure 6. An Intelligent System that selects the most reliable AI response

Figure 6 shows a simple example of an intelligent system that combines two different AI engines that are used to visually recognize identical objects. Reliability might depend upon illumination and weather. Figure 5 and Figure 6 both show the importance of calculating reliability when operating intelligent systems.

V. Designing Intelligent Systems

Intelligent systems rely on the capability of measuring the quality of knowledge [46]. To this purpose, it is necessary to consider how software can be measured. This is not so easy, as software is not a tangible entity. The standard solution to this measurement challenge is to measure the functionality of software. As always in measurement theory, this is best achieved by constructing a model for the functionality and measuring the relevant model elements.

A. The COSMIC Model for Functionality

The COSMIC standard identifies layers. The layers' boundaries detect the flow of data moving from one object into another. Every *Data Movement* transports a *Data Group*, identifying the data moved from one object to another.

Data groups hold the information needed to assess privacy protection needs, or safety risk exposure, of data. They also transport knowledge from one *Object of Interest* into another. In certain cases, the data groups contain enough information to allow for generating code out of a COSMIC model [47].

The constituent element of the COSMIC model is a *Functional Process*. A functional process is an object together with a set of data movements, classified as either **R**ead or **W**rite, or **E**ntry or e**X**it. These data movements connect the functional process with *Persistent Data Stores*, or *Devices* respectively *Other Applications*, e.g., an AI engine. They

represent the *Functional User Requirements* (FUR) for the software being measured [48, p. 42].

Figure 7 is a sample *Data Movement Maps* according to ISO/IEC 19761 COSMIC. The connectors represent *Data Movements*. The data groups that they convey can be viewed as a single data set. Each data group should occur in a model only once. Its uniqueness is indicated by color-filled trapezes at their origin. Another move of the same data group between the same objects within a COSMIC functional process lets the trapeze blank.

Objects of Interest: For data movement maps, we draw four types of lifelines:

- **Functional Processes**: Objects that perform one or several functional processes in the COSMIC sense.
- **Persistent Data Store**: Objects that persistently hold data.
- **Devices**: A device can be a system user or anything providing or consuming data. They send (Entry) or receive (eXit) data groups from or to functional processes.
- **Other Applications**: Other applications use functional processes the same way as devices do; however, they typically represent other software or systems that can be modeled the same way using data movement maps.

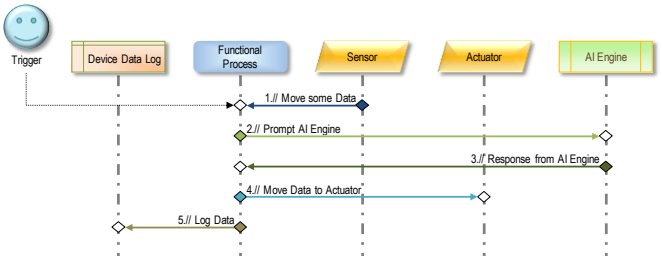

Figure 7. Sample Data Movement Map

The lifeline for functional processes represents, for example, a *Virtual Machine* (VM) or an *Electronic Control Unit* (ECU) that performs various calculations and implements several functional processes as defined in the COSMIC manual [48, p. 42]. *Triggers* usually indicate the starting data movement of one COSMIC functional process. Thus, one functional process lifeline can have several triggers. Lifelines representing persistent data stores can provide data services for more than one functional process. LLMs, or other AI engines appear as *Another Application* in the data movement maps.

B. Assigning Reliability Scores to Objects of Interest

A typical data movement map for an intelligent system consists of devices such as sensors, actuators, and persistent data sources, collecting data and delivering them to a functional process that prepares them as input to a suitable AI engine such as a neural network, a specific knowledge base, or some search engine. Another functional process will then work with its response and execute recommendations using a *Generative Pretrained Translator* (GPT) to communicate the response to a user through an output device. This is necessary because all knowledge in intelligent systems is represented by token vectors.

However, neither sensors, actuators nor an AI tool can be trusted 100%. Thus, data groups need to have special attribute for this: *Reliability*.

If data is persistently stored, it retains its reliability. All other data has some degree of reliability that is either known from physical devices, by assessing an AI tool with a test set, or by the functional processes the data groups go through. Reliability originates from devices or other applications. Although reliability is measured commonly by percentage numbers, it is a standard deviation, not a linear reliability average [49].

C. Combining Reliabilities with Regard to FUR

The data movement map describes how data groups move through the software. The functional processes combine these reliabilities by combining uncertainties associated with knowledge-based actions. In our example, the reliability of the data groups originating from the functional process in Figure 7 is the combination of the reliability of all incoming data. The expected overall reliability of the intelligent system is calculated from the uncertainty expectations of the output of the functional process.

The way data is combined in functional processes depends on the FUR that implements them. This requires an understanding of the discipline of requirements engineering for knowledge-processing systems [50]. As explained before, requirements address knowledge with distinct levels of certainty: While observations are usually not completely certain and learning concepts will never be fully reliable, there are rules, the so-called *Lambda Concepts*, that work in a mechanical way, preserving reliability. It is not a promising idea to implement Lambda concepts by arrow schemes. There exist much more efficient tools: conventional programming, that in turn also have a model expressible as arrow schemes. Typically, in intelligent systems Lambda concepts are implemented as programs according to rules, mostly in Python [51].

Depending on the FUR, functional processes can also reduce uncertainty, for instance when it selects the most reliable response between various kinds of AI engines as shown in Figure 6. The diverse ways of combining reliability are discussed in more detail in [46].

D. Propagating Reliability through Functional Processes

Lambda concepts might extract one source and discard the second or do substitution. Such operations might keep uncertainty unchanged by a functional process. Functional processes that implement Lambda concepts combine according to the Max principle, which preserves the maximum reliability of the different input data groups and propagates the maximum degree of knowledge reliability, while the normal combination of different uncertainties usually increases uncertainty and is called the $Comb$ principle. Since subsequent uncertainties can correct the initial uncertainty of data groups from the first source, the reliability is expected to decrease only according to the expected value obtained with the statistical sensitivity analysis, which means that the reliability decreases at a lower rate than with naive multiplication.

Let u_i be the uncertainty of the i^{th} input in a functional process with n input data groups. The uncertainty of the output of this functional process is:

$$u_{Comb} = \sqrt{\sum_{i=1}^{n} u_i^2} \quad (18)$$

This is the Euclidean distance between the uncertainties of the input data groups. Thus, the reliability propagation follows the same statistical rules as the profiles in Six Sigma transfer functions [33, p. 34].

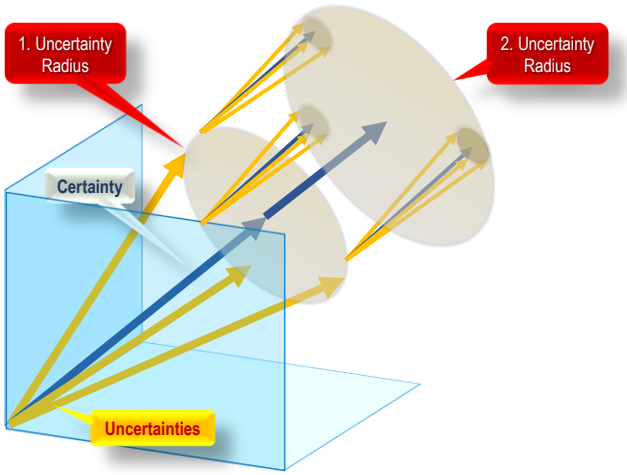

Figure 8. Combination of Uncertainties originating from AI input

The *Reliability* of a functional process with n input data movements carrying data groups with uncertainties $u_1, u_2, \dots, u_n$ is defined as:

$$Reliability(u_1, u_2, \dots, u_n) = 1 - u_{Comb} \quad (19)$$

With graphical visualization (Figure 8) we aim to explain the statistical methods used without going into formalisms. The figure proofs equation (19) in the special case of three dimensions ($n = 3$).

Assume there are some uncertainties originating from the first input source. The response range in the n-dimensional space of all responses is produced by some functional process. By combining this with the uncertainty originating from the second input source, it is unknown where the second uncertainty builds over the previous partial response.

Thus, the bundle of outcomes with an encompassing second uncertainty radius representing the expected uncertainty of a functional process with two inputs combined can correct part of the first. Expected uncertainties thus must be combined by using equation (19).

Reliability of a data group might change when originating from different functional processes. Also, a functional process might produce more than one data group as an eXit or Write data movement with different reliabilities, dependent from the data group.

E. Making an LLM Reliable

Not if we combine with a source of information with known reliability. We should set up a functional process connecting The LLM with some factual repository such as Wolfram|Alpha, or whatever is suitable for its topics. Best we feed the facts to the LLM and let the LLM apply its pretrained conversational capabilities as a transformer to transfer factual knowledge into arguments and explanations. Then the reliability combines from the reliability of the facts with the reliability of the LLM as a transformer and we can chain it with a Lambda concept that checks whether the LLM has kept well to the original facts. Thus, the FUR we have against the LLM is that we expect it to reproduce available facts with a known reliability. Here we assume 95%. As a further assumption, Wolfram|Alpha has been measured to be 98% reliable in the chosen context.

In Figure 9, there are two functional processes, one feeding the facts to the LLM and one comparing results. The persistent store serves for logging results, learning, and for communication between the two functional processes. The data movement map explains how the data groups are moved from one object to another.

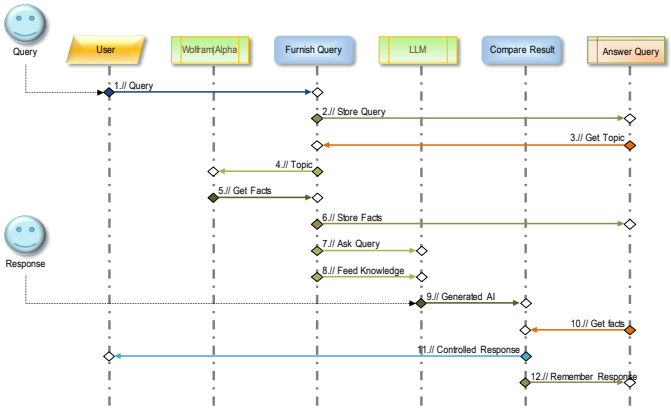

Figure 9. Data Movement Map for Combining LLM with Wolfram|Alpha

The *Compare Result* functional process in Figure 8 combines uncertainties according to the statistical methods explained in equations (18) and (19) as follows:

$$\sqrt{5\%^2 + 2\%^2} = 5.4\% \quad (20)$$

With regard to equation (19), this yields a reliability of 94.6% for the query functionality represented in Figure 9. Figure 9 and the equation (20) have been created and computed using an Excel-based tool from the authors, which is available to interested reader [33].

F. The Future of AI: Intelligent Systems

The current hype with AI is suffering from the same problem earlier attempts had: QFD, Expert Systems, and many other machine-based reasoning and decision tools could not explain how reliable they are. You could believe them or not; and sometimes, the non-believers proved to be true.

The new ability to build LLMs with recognizable CoT still does not answer the question, how reliable they are. Taking the graph model as an explanation of reliability, recognizing that knowledge and algorithmic processing are from the same source, namely the graph model, shows a way how to build intelligent systems with known reliability. It makes AI-based decisions and process control suitable for legal assessment and technical certification.

It is obvious that the reliability of a data group might change during operations of an intelligent system. Functional processes can calculate reliability, keep a log trace of it, and use it to guide processing through all programming steps in the data movement map.

In theory, intelligent systems consist of a controlling combinator, in most cases realized by implementing some functional processes, and an ANN part, typically implementing an LLM that is trained on the specific knowledge domain. This reflects equations (16) and (17). Obviously, both parts, the controlling combinator and the knowledge acquisition combinator. Both can be described as arrow schemes in the graph model, but they are implemented differently, in the most effective way. It is indeed not necessary to train an LLM to do reasoning, because a controlling combinator implementing functional processes in Python are much more effective. Explaining, maintaining, and improving such a controlling combinator is much easier in Python (or any other suitable programming environment) than training an LLM. However, it is possible to train an LLM in logical reasoning, but this is not needed. It is much more straightforward, and way more effective, to use Engeler's controlling combinator as a design paradigm for intelligent systems.

VI. Conclusion and Future Work

Like humans, the result of any ANN is as unreliable as any result from an uneducated NNN. Without logical foundations, logical reasoning, and feedback from the environment, humans are also just hallucinating.

Intelligent systems can do better, except getting feedback and learning from it. The key is to combine combinators representing neural networks with combinators doing logical derivations. This is primarily a design principle, but secondly also an operating paradigm.

In this paper, we have provided evidence for:

- RQ 1: DNNs can be represented in the graph model of combinatory logic as well as any other neural network, including the brain or QFD;
- RQ 2: CoT does not relate to a defined and unique sequence of arrow schemes because of missing normal form, but can be explained using other arrow schemes, such as QFD;
- RQ 3: Intelligent systems explain how AI behavior can be controlled.

The graph model of combinatorial logic does not provide an alternative for implementing AI, but it is an excellent guide and theoretical foundation for what can be done with AI, for explaining AI, but also for learning where AI meets its limits. The current step forward is collecting several designs of intelligent systems with controlling combinators, finding methods for measuring reliability and defining suitable convergence gaps. This work in progress of the authors is shared with interested parties; the authors have no institution or sponsor to help with this [52].

It is possible that ANNs can learn logical reasoning based on facts. However, combining AI engines with feedback loops originating from reality requires much less effort. Testing AI results for feasibility and physical soundness using traditional programming methods creates trustworthiness and adds credibility and explainability to AI results.

It remains the idea that AI could be explained by searching for arrow schemes that provide the same responses. Since combinatory logic does not have normal forms, this seems feasible. It could be used as a validation process for AI. However, for now, this is a future research project.

Acknowledgment

The authors would like to thank Lab42 in Davos for asking excellent questions and promoting the ARC Challenge [10], now ARC Prize [53], and to the anonymous reviewers who helped to improve this paper. Special thanks to Erwin Engeler for his suggestions and availability for valuable discussions with his former student.

References

[1] T. M. Fehlmann and E. Kranich, "The Graph Model of Combinatory Logic as a Model for Explainability," in *ThinkMind Digital Library https://www.thinkmind.org*, Valencia, Spain, Proc. Int. Conf. Systems Explainability (EXPLAINABILITY 2024), pp. 40-46, 2024..

[2] K. Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I," *Monatshefte für Mathematik und Physik,* vol. 38, no. 1, pp. 173-198, 1931.

[3] A. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society,* vol. 42, no. 2, pp. 230-265, 1937.

[4] P. Raatikainen, "Gödel's Incompleteness Theorems," in *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed., 2020.

[5] H. Curry and R. Feys, Combinatory Logic, Vol. I, Amsterdam: North-Holland, 1958.

[6] A. Church, "The Calculi Of Lambda Conversion," *Annals Of Mathematical Studies 6,* 1941.

[7] E. Engeler, "Algebras and Combinators," *Algebra Universalis,* vol. 13, pp. 389-392, 1981.

[8] F. Rosenblatt, "The Perceptron: A Perceiving and Recognizing Automaton (Project PARA)," Cornell Aeronautical Laboratory, Inc., Buffalo, 1957.

[9] M. Minsky and S. Papert, Perceptrons: An Introduction to Computational Geometry, 2nd edition with corrections ed., Cambridge, MA: The MIT Press, 1972.

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

[11] K. Wijekumar, B. J. Meyer, and P. Lei, "High-fidelity implementation of web-based intelligent tutoring system improves fourth and fifth graders content area reading comprehension," *Computers & Education,* vol. 68, pp. 366-379, 2013.

[12] A. Peterson, "Literacy is More than Just Reading and Writing," National Council of Teachers of English, 23 March 2020. [Online]. Available: https://ncte.org/blog/2020/03/literacy-just-reading-writing/. [Accessed 14 May 2025].

[13] M. v. Gerven and S. Bohte, "Artificial Neural Networks as Models of Neural Information Processing," in *Frontiers in Computational Neuroscience*, Lausanne, 2017.

[14] IBM TechXchange Community, "What is explainable AI?," IBM Reserach, Available at: https://www.ibm.com/topics/explainable-ai. [Accessed 15 May 2025].

[15] F. Dallanoce, "Explainable AI: A Comprehensive Review of the Main Methods," Medium.com, San Francisco, California, 2022.

[16] M. R. Morris, J. Sohl-Dickstein, N. Fiedel, T. Warkentin, and A. Dafoe, "Levels of AGI for Operationalizing Progress on the Path to AGI," arXiv:2311.02462v4 [cs.AI], Cornell University, 2024.

[17] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," arXiv:2201.11903v6 [cs.CL], Cornell University, 2023.

[18] S. Xu, W. Xie, L. Zhao, and P. He, "Chain of Draft: Thinking Faster by Writing Less," arXiv:2502.18600v2, Cornell University, Ithaca, NY, 2025.

[19] H. Curry, J. Hindley, and J. Seldin, Combinatory Logic, Vol. II, Amsterdam: North-Holland, 1972.

[20] T. M. Fehlmann and E. Kranich, "Intuitionism and Computer Science – Why Computer Scientists do not Like the Axiom of Choice," *Athens Journal of Sciences,* vol. 7, no. 3, pp. 143-158, 2020.

[21] T. M. Fehlmann and E. Kranich, "A General Model for Representing Knowledge - Intelligent Systems Using Concepts," *Athens Journal of Sciences,* vol. 11, pp. 1-18, 2024.

[22] H. Barendregt and E. Barendsen, Introduction to Lambda Calculus, Nijmegen: University Nijmegen, 2000.

[23] H. P. Barendregt, "The Type-Free Lambda-Calculus," in *Handbook of Math. Logic*, vol. 90, J. Barwise, Ed., Amsterdam, North Holland, 1977, pp. 1091 -1132.

[24] T. M. Fehlmann and E. Kranich, "The Fixpoint Combinator in Combinatory Logic - A Step towards Autonomous Real-time Testing of Software?," *Athens Journal of Sciences,* vol. 9, no. 1, pp. 47-64, 2022.

[25] E. Zachos, "Kombinatorische Logik und S-Terme," ETH Dissertation 6214, Zurich, 1978.

[26] Lisp-Community, "Common Lisp," 2015. [Online]. Available: https://lisp-lang.org/. [Accessed 15 May 2025].

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

[28] V. Zhong, J. Mu, L. Zettlemoyer, E. Grefenstette, and T. Rocktäschel, "Improving Policy Learning via Language Dynamics Distillation," arXiv:2210.00066v1, Cornell University, 2022.

[29] E. Engeler, "Neural algebra on "how does the brain think?"," *Theoretical Computer Science,* vol. 777, pp. 296-307, 2019.

[30] P. Barceló, J. Pérez, and J. Marinković, "On the Turing Completeness of Modern Neural Network Architectures," arXiv: 1901.03429v1 [cs.LG], Cornell University, Ithaca, NY, 2019.

[31] G. Hinton, "A Practical Guide to Training Restricted Boltzmann Machines," in *Neural Networks: Tricks of the Trade*, Springer Lecture Notes in Computer Science, vol 7700. Berlin, Heidelberg, 2012, pp. 599-619.

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

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

[34] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation,* vol. 19, pp. 297-301, 17 August 1964.

[35] T. M. Fehlmann and E. Kranich, "The World Formula and the Theorem of Perron-Frobenius: How to Solve (Almost All) Problems of the World," *Athens Journal of Sciences,* vol. 10, no. 2, pp. 95-110, 2023.

[36] T. L. Saaty and J. M. Alexander, Conflict Resolution: The Analytic Hierarchy Process, New York, NY: Praeger, Santa Barbara, CA, 1989.

[37] Ł. Lachowski, "On the Complexity of the Standard Translation of Lambda Calculus into Combinatory Logic," *Reports on Mathematical Logic,* vol. 53, pp. 19-42, 2018.

[38] Z. Jin and W. Lu, "Self-Harmonized Chain of Thought," arXiv:2409.04057v1 [cs.CL], Cornell University, 2024.

[39] T. M. Fehlmann and E. Kranich, "How to Explain Artificial Intelligence to Humans - Learning from Quality Function Deployment," in *EuroSPI*, Munich, 2024.

[40] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," arXiv:2106.09685v2 [cs.CL], Cornell University, Ithaca, NY, 2021.

[41] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-Augmented Generation for Large Language Models: A Survey," arXiv:2312.10997v5 [cs.CL] , Cornell University, 2024.

[42] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," arXiv:2005.11401v4 [cs.CL], Cornell University, 2021.

[43] T. M. Fehlmann and E. Kranich, "Making Artificial Intelligence Intelligent - Solving the Control Problem for Artificial Neural Networks by Empirical Methods," in *Human Systems Engineering and Design (IHSED2024): Future Trends and Applications. AHFE (2024) International Conference*, Split, 2024.

[44] M. Ganaie, M. Hu, A. Malik, M. Tanveer and P. Suganthan, "Ensemble deep learning: A review," arXiv:2104.02395 [cs.LG], Cornell University, Ithaca, NY, 2022.

[45] DeepSeek-AI, "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning," arXiv:2501.12948v1, Cornell University, Ithaca, NY, 2025.

[46] T. M. Fehlmann and E. Kranich, "Measuring the Quality of Intelligent Systems," in *Intelligent Systems and Applications*, vol. 1066, K. Arai, Ed., IntelliSys 2024, Amsterdam, Lecture Notes in Networks and Systems, Springer, Cham, 2024, pp. 438-455.

[47] A. Oriou, E. Bronca, B. Bouzid, O. Guetta, and K. Guillard, "Manage the automotive embedded software development with automation of COSMIC," in *IWSM Mensura 2014*, Rotterdam, 2014.

[48] COSMIC Measurement Practices Committee, "The COSMIC Functional Size Measurement Method – Version 4.0.2 – Measurement Manual," The COSMIC Consortium, Montréal, 2017.

[49] L. S. Jutte, K. L. Knight, B. C. Long, J. R. Hawkins, S. S. Schulthies, and E. B. Dalley, "The Uncertainty (Validity and Reliability) of Three Electrothermometers in Therapeutic Modality Research," *Journal of Athletic Training,* vol. 40, no. 3, pp. 207-210, 2005.

[50] T. M. Fehlmann and E. Kranich, Requirements Engineering for Cyber-Physical Products, Systems, Software and Services Process Improvement. EuroSPI 2023 ed., Vols. Systems, Software and Services Process Improvement. EuroSPI 2023, M. Yilmaz, C. P. A. Riel and R. Messnarz, Eds., Grenoble: Communications in Computer and Information Science, Springer, Cham, 2023.

[51] S. F. Python, "Python Setup and Usage," 2001-2024. [Online]. Available: https://docs.python.org/3/using/index.html. [Accessed 27 April 2024].

[52] T. M. Fehlmann, "Intelligent Systems - A Recollection," Euro Project Office AG, 9 May 2024. [Online]. Available: https://web.tresorit.com/l/AXX78#FaBkGqfY2cF_JsVmX70_ng.

[53] Lab 42, "ARC Price," Mike Knoop, Davos and San Francisco, 2024.