Introducing Neural Mesh for Logical Synthesis Models

Péter Baranyi, and Ádám B. Csapo

Abstract—**The paper introduces the Neural Mesh, a novel architecture that serves as an alternative to conventional neural network models widely used in contemporary AI systems. In parallel, the paper introduces the term Logical Synthesis Model (LSM) as a complement to Large Language Models (LLMs). While LLMs primarily rely on statistical representations and language-based reasoning to acquire and generate knowledge, Neural Mesh-based LSMs focus on structured representations and engineering reasoning to represent, analyze, synthesize, and control systems. LLMs excel at learning and reasoning over human knowledge expressed through language, whereas LSMs aim to learn structured representations that support the "understanding" of physical systems. In this sense, LLMs and LSMs may play roles analogous to the cerebrum and the cerebellum in biological intelligence. The novelty of the paper lies in bridging and combining the concepts of TP models, TP model transformation, and neural-network architectures. The proposed Neural Mesh is mathematically equivalent to the well-established TP model function family. The TP model transformation provides a framework for constructing TP models with unique features that are particularly advantageous for systems and control applications. The contribution of the Neural Mesh is that it represents these unique features in the form of a special neural-network architecture, where these features are expressed as trainable neural connections and parameters. As a result, elements that are traditionally determined through an offline TP model transformation become directly tunable through learning, while preserving the mathematical structure and control-theoretic advantages of the TP model framework. Therefore, the Neural Mesh representation opens a future research direction in which the TP model transformation is effectively embedded into the training process itself. Rather than generating a TP model through a subsequent offline transformation, the corresponding TP-model components are directly constructed and tuned during system identification via neural-network training tools.**

I. INTRODUCTION

One of the fundamental limitations of neural-network-based system modelling – particularly in the case of dynamic system control – is that the resulting neural network obtained through system identification is generally not represented in a mathematical form that can be directly utilized by mainstream optimization and control-design methodologies. In dynamic system modelling and control, the mathematical representation of the model is of primary importance, as it determines whether established control theories can be interpreted and methods for controller synthesis, stability analysis, robustness verification, and performance optimization can be directly applied. Consequently, conventional neural networks remain largely black-box models in the context of system modelling and control, since the mathematical structures and control-theoretic properties embedded in the learned representation are generally not explicitly accessible for subsequent analysis, verification, and controller synthesis.

To address these limitations, the paper introduces the term Logical Synthesis Model (LSM), inspired by the notion of Large Language Models (LLMs). Logical Synthesis Models (LSMs) differ fundamentally from Large Language Models (LLMs). LLMs are primarily designed for knowledge representation and content generation within the domain of Generative AI, relying on statistical representations, language-based reasoning, and prompting techniques. In contrast, LSMs are intended for system representation within the domain of Engineering AI. Rather than modelling linguistic knowledge, LSMs learn structured representations of systems that support engineering reasoning, enabling subsequent analysis, synthesis, optimization, verification, and control design.

The LSM concept substantially constrains the structure of the Neural Mesh, as the architecture must be designed to represent the dynamics of physical systems as faithfully as possible. As a result, the Neural Mesh is naturally driven by the mathematical structure of physical systems and engineering models, rather than by the statistical language representations that characterize LLMs. Consequently, the Neural Mesh follows a considerably more structured architecture than conventional neural networks. The primary objective of training is not to discover the network structure itself, but rather to tune the parameters associated with a predefined mathematical representation implemented by the architecture. In this sense, learning in a Neural Mesh focuses predominantly on identifying the coefficients of the underlying mathematical model. This contrasts with many general-purpose neural network architectures, where both the network parameters and, in many cases, the network topology, layer composition, and structural configuration may also be subject to optimization.

An interesting observation is that the distinction between LLMs and LSMs exhibits a loose analogy to the relationship between the cerebrum and the cerebellum in biological intelligence. While the cerebrum is primarily associated with knowledge representation, abstraction, and high-level reasoning, the cerebellum operates in close interaction with sensory signals and the physical environment, contributing to the interpretation, coordination, and control of physical processes.

A further interesting parallel is that the cerebellum possesses

P. Baranyi (peter.baranyi@uni-corvinus.hu) and Á. B. Csapó are with the Corvinus Institute for Advanced Studies (CIAS) and Institute of Data Analytics and Information Systems at the Corvinus University of Budapest, Budapest, Hungary and the Hungarian Research Network (HUN-REN). The authors declare that there is no conflict of interest regarding the publication of this paper. The research work was supported by National Research, Development and Innovation Office HURIZONT PROJECT 2025- 2026 NKFIH - 2024-1.2.3-HU-RIZONT-2024-00030.

DOI: 10.36244/ICJ.2026.2.3

a remarkably regular and highly structured architecture. It is organized into a relatively small number of well-defined layers, and learning is believed to occur primarily through the adaptation of synaptic connections rather than through changes in the overall architectural organization. Similarly, the Neural Mesh adopts a largely predefined structure motivated by the requirements of system representation, while learning is primarily associated with the adjustment of the parameters embedded within this structure.

A. Novel contribution

The paper introduces the Neural Mesh, whose mathematical transfer function is identical to one of the most widely used polytopic system representations in modern control theory. More specifically, the Neural Mesh and higher-order Tensor Product (TP)-based polytopic models [1]–[10] share exactly the same mathematical formulations. Consequently, the extensive body of control design methodologies developed for TP model transformation, TP-based polytopic representations, polytopic models in general, and Takagi–Sugeno (TS) fuzzy model representations can be directly applied to Neural Mesh models.

This connection provides immediate access to a broad range of established analysis, verification, and controller synthesis techniques. In particular, the Parallel Distributed Compensation (PDC) framework [11] becomes directly applicable to Neural Mesh representations. Since PDC-based controller synthesis can be formulated through Linear Matrix Inequalities (LMIs) [12], one of the most extensively investigated optimization frameworks in modern control theory, the Neural Mesh naturally inherits a rich set of mathematically rigorous tools for stability analysis, performance optimization, robustness verification, and controller design.

A natural question arises: if the transfer function realized by the Neural Mesh is mathematically identical to the model obtained through Tensor Product (TP) model transformation, which itself is based on a Higher-Order Singular Value Decomposition (HOSVD) structure and various convex-hull representations, why is it worthwhile to introduce and study the same mathematical representation in the form of a Neural Mesh?

The answer lies in the fundamentally different perspectives of the TP model, TP model transformation and Neural Mesh. The TP model transformation determines which components of the TP model should be constructed and how they should be configured in order to obtain advantageous properties from the perspective of systems and control theory (this is the reason why the general TP model form is popular in system theories). The Neural Mesh, in turn, represents the TP model in a neural-network structure in which these components are reduced to standard neural connections and trainable parameters commonly used in neural networks. Consequently, the determination of TP model components is no longer performed solely through an offline transformation procedure but can be achieved through a tunable learning process.

In this sense, the Neural Mesh transforms TP model construction and optimization from an offline execution paradigm into a trainable neural-learning paradigm while preserving the underlying mathematical and control representation. This representation also makes the individual components responsible for key TP-model properties – such as convex-hull manipulation – which reduce to simple weighting-matrix operations, explicitly identifiable within the neural architecture. As a result, these components become directly accessible to training and optimization procedures developed for neural networks, while simultaneously retaining the theoretical advantages and interpretability offered by the TP model transformation framework.

Therefore, an important aspect is that the Neural Mesh formulation makes the role of convex-hull optimization during learning more explicit. This is significant because a recently (2026) established formal mathematical result shows that one of the most critical, if not the most critical, elements of polytopic PDC design is the determination of the optimal convex hull [10]. This observation challenges the common view in the literature that optimality can be considered primarily at the level of the controller design once a polytopic model has been fixed. The influence of convex hull manipulation in TP model forms has already been observed in some engineering control-design applications, where different convex hull structures often lead to significantly different design outcomes [13]–[32].

B. Opening future research directions

From the above perspectives, the objective of learning in Neural Mesh models is not limited to achieving modelling accuracy. Rather, learning may also be interpreted as the search for an optimal convex-hull representation that supports subsequent analysis, verification, and controller synthesis. In the Neural Mesh architecture, this convex-hull structure appears as an explicit and visible structural component, thereby opening a new research direction on how the model can be trained such that the learning process itself leads to convex-hull optimization.

Conversely, this interpretation also opens another future research direction: operations and transformations between Neural Mesh models may be formulated as learnable procedures corresponding to transformations and operations between TP-based polytopic models. In this way, the Neural Mesh provides not only a learnable representation of TP-type polytopic models, but also a possible learning-based framework for carrying out transformations between such representations published in [5], [7]–[9].

In summary, although the transfer function of the Neural Mesh is mathematically identical to the representation obtained through TP model transformation, the Neural Mesh offers a substantially different perspective on the same underlying mathematical structure. Its architecture exposes structural components that can be more naturally connected to learning and training methodologies developed for neural networks, while simultaneously making explicit those elements that are of particular importance from a control-theoretic viewpoint.

II. NOTATION

The following notation is used in the paper:

- Indices are denoted by $n, r, c, j, l \ldots$ with corresponding upper bounds denoted as $N, R, C, J, L \ldots$ respectively.
- Scalars, vectors, matrices and tensors (multidimensional matrices) are denoted as $b \in \mathbb{R}$, $\mathbf{b} \in \mathbb{R}^{J}$, $\mathbf{B} \in \mathbb{R}^{J_1 \times J_2}$, $\mathcal{B} \in \mathbb{R}^{J^O}$. E.g. notation $\mathcal{B} \in \mathbb{R}^{C^N \times J^O}$ is equivalent to $\mathcal{B} \in \mathbb{R}^{C_1 \times \ldots \times C_N \times J_1 \times \ldots \times J_O}$.
- $[\mathcal{B}]_{index}$ addresses elements, e.g. $[\mathcal{B}]_{c_1,c_2,\ldots c_N} = \mathbf{B}_{c_1,c_2,\ldots c_N} \in \mathbb{R}^{J^2}$ of $\mathcal{B} \in \mathbb{R}^{C^N \times J^2}$;
- $\Omega \subset \mathbb{R}^N$ denotes a domain defined as the Cartesian product of one-dimensional intervals $\omega_n = [\omega_n^{min}, \omega_n^{max}] \subset \mathbb{R}$, i.e., $\Omega = \omega_1 \times \ldots \times \omega_N$.
- $\bigboxtimes_{n=1}^{N}$ denotes the vector(matrix)-tensor product, that is defined as

$$\mathcal{B} \bigboxtimes_{n=1}^{N} \mathbf{f}_n(x_n) = \tag{1}$$

$$= \sum_{c_1=1}^{C_1} \sum_{c_2=1}^{C_2} \cdots \sum_{c_N=1}^{C_N} \prod_{n=1}^{N} [\mathbf{f}_n(x_n)]_{c_n} [\mathcal{B}]_{c_1,c_2,\ldots c_N}.$$

III. Neural Mesh Architecture

The Neural Mesh maps an input vector $\mathbf{x} = [x_1, \ldots, x_N] \in \Omega \subset \mathbb{R}^N$ through four stages: (1) a per-dimension *Resolution Layer* composed of triangular activation functions that define subregions of the input domains; (2) a per-dimension *Complexity Layer* that classifies the input values through meaningful weightings – similarly to fuzzy membership functions – while characterizing the rank of the Neural Mesh; (3) a *Mesh Layer* (or *Core Mesh*) that forms the mesh (tensor-product) combination of the per-dimension outputs of the Complexity Layer; and (4) an *Output Layer* that weights and aggregates the outputs of the Mesh layer.

A. Resolution Layer

The Resolution Layer, i.e. the input layer, consists of R neurons, where R defines the "Resolution" of the representation. As R increases, the sensitivity – or equivalently, the resolution – with respect to the input also increases, see Figure 1.

Each neuron has one input x and one output s_r. Each neuron senses one subregion of the input space. The activation function of a neuron defines the subregion of the input space in which that neuron is active. Therefore, each neuron has an offset input g_r, and to ensure the systematic overlapping between subregions of the neurons, neighbouring neurons share their offset information, thereby enforcing a consistent boundary between their respective activation domains.

Consequently, the combination of the neuron-specific offsets and activation functions as shown on the upper plot in Figure 1 – induces a partitioning of the input space into subregions. Each subregion is assigned to the neighbouring neurons as depicted in Figure 1.

The activation function – that is, the neuron output – is denoted by $s_r(x, g_r)$, where "s" refers to "smoothing". In the present case, piecewise linear activation functions are applied; however, bell-shaped or various higher-order functions can also be employed. Therefore the activation functions $s_r(x, g_r)$ contribute to the smoothness of the output generated by the entire Neural Mesh.

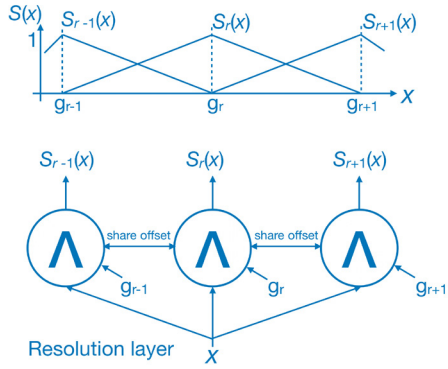

Fig. 1: Resolution Layer: Input Neuron and the activation functions

The output of the Resolution Layer is stored in a vector $\mathbf{s}(x, \mathbf{g}) \in \mathbb{R}^R$ with offset vector $\mathbf{g} = \begin{bmatrix} g_1 & \ldots & g_R \end{bmatrix} \in \mathbb{R}^R$, where $g_r < g_{r+1}$, as

$$\mathbf{s}(x, \mathbf{g}) = \begin{bmatrix} s_1(x, \mathbf{g}) & s_2(x, \mathbf{g}) & \ldots & s_R(x, \mathbf{g}) \end{bmatrix} \tag{2}$$

such that if $x \in [g_r, g_{r+1}]$ then

$$s_r(x, \mathbf{g}) = 1 - \frac{x - g_r}{g_{r+1} - g_r} \text{ and } s_{r+1}(x, \mathbf{g}) = 1 - s_r(x, \mathbf{g}), \tag{3}$$

and all other $s_z(x, \mathbf{g}) = 0$ ($z = 1, ..., R$ and $\neq r, r+1$), see the upper plot in Figure 1.

When the Neural Mesh has N inputs x_n, each input is associated with a Resolution Layer containing R_n neurons. The outputs of the Resolution Layers are:

$$\mathbf{s}_n(x, \mathbf{g}_n) = \begin{bmatrix} s_{n,1}(x_n, \mathbf{g}_n) & \ldots & s_{n,R_n}(x_n, \mathbf{g}_n) \end{bmatrix}, \tag{4}$$

where the offset is defined for each input by vector $\mathbf{g}_n = \begin{bmatrix} g_{n,1} & \ldots & g_{n,R_n} \end{bmatrix}$.

Remark 1: Note that at most two neighbouring neurons per input are active for any input x_n. Thus, the computation is reduced to these two neurons.

B. Output Layer

Each output y_l of the Output Layer is produced by a single additive neuron that has no activation function. Each neuron generates a weighted sum of the outputs of the previous layer, see Figure 2. Let the output of the previous layer be stored in the vector $\mathbf{p} \in \mathbb{R}^M$. Then, the activation at the Output Layer is given by

$$y_l = \mathbf{p}^T \mathbf{b}_l,$$

where the vector $\mathbf{b}_l = \begin{bmatrix} b_{l,1} & b_{l,2} & \ldots & b_{l,M} \end{bmatrix}^T \in \mathbb{R}^M$ contains the gains b_m of the connections between the previous layer and the neuron in the Output Layer, see Figure 2.

If the output has a higher-order structure, e.g. the output values are stored in a tensor structure $\mathcal{Y} \in \mathbb{R}^{J^O}$, that means that the linear index "l" is replaced with multi-linear indexing $j_1, j_2, \ldots j_O$ – the output is determined as

$$[\mathcal{Y}]_{j_1,j_2,\ldots j_O} = \mathbf{p}^T \mathbf{b}_{j_1,j_2,\ldots j_O}. \tag{5}$$

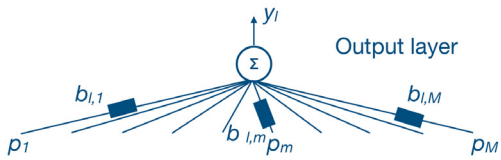

Fig. 2: Output Layer

This can be given by tensor-vector product as

$$\mathcal{Y} = \mathcal{B} \times_1 \mathbf{p}, \tag{6}$$

where the tensor of the gains is constructed as $\mathcal{B} \in \mathbb{R}^{M \times J^O}$, e.g. it contains vectors $\mathbf{b}_{j_1,j_2,\ldots j_O} \in \mathbb{R}^M$.

C. Complexity Layer

The Internal Complexity Layer is constructed by extending the Resolution Layer with an Output Layer, see Figure 3. It composes a higher-level information representation of the inputs. The Complexity Layer has C neurons and outputs. It characterizes how complex or simple the contribution of the underlying input is to the output of the Neural Mesh. Mathematically, in tensor algebraic terms, this corresponds to the rank of the input representation. Specifically, the number of neurons in the Complexity Layer can express the input rank of the Neural Mesh. If C is larger than the rank of the information associated with this input, then redundant neurons appear in the Neural Mesh. If C is smaller than the rank, the Neural Mesh cannot achieve the best possible approximation. Therefore, during training, C is increased up to at most the rank, with the objective of finding an appropriate balance between the number of neurons and the approximation accuracy of the Neural Mesh. This can also be associated with the micro or macro level of the input-output mapping of the Neural Mesh, or can be interpreted or readily linked with principal component analysis and singular functions. A further important point is that the functions implemented by the individual neurons can be trained in such a way that they represent the input information in a meaningful and interpretable form, similarly to fuzzy sets, see Section III/F.

The overall transfer function of the Resolution Layer and Complexity Layer is

$$f_c^C(x) = \sum_{r=1}^{R} s_r(x, \mathbf{g}) u_{r,c} = \mathbf{s}(x, \mathbf{g}) \mathbf{u}_c, \tag{7}$$

which leads to

$$\mathbf{f}^C(x) = \begin{bmatrix} f_1^C(x) & \ldots & f_C^C(x) \end{bmatrix} = \mathbf{s}(x, \mathbf{g}) \mathbf{U}, \tag{8}$$

where matrix $\mathbf{U} \in \mathbb{R}^{R \times C}$ contains the connection gains $u_{r,c}$.

When the Neural Mesh has N inputs x_n, each input is transferred through the Resolution Layer and the Complexity Layer as

$$\begin{aligned} \mathbf{f}_n^C(x_n) &= \begin{bmatrix} f_{n,1}^C & (x_n) & \ldots & f_{n,C_n}^C(x_n) \end{bmatrix} \\ &= \mathbf{s}_n(x_n, \mathbf{g}_n) \mathbf{U}_n. \end{aligned} \tag{9}$$

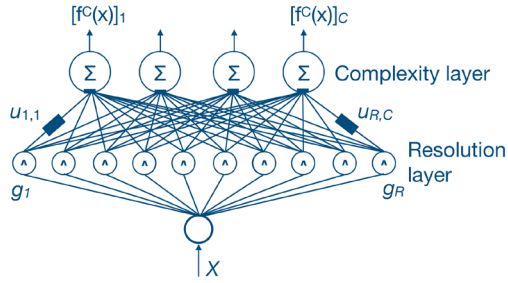

Fig. 3: Complexity Layer

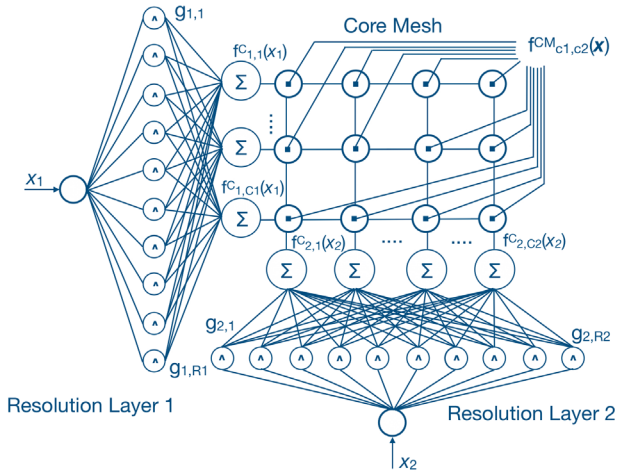

Fig. 4: Two-input Core Mesh

D. Core Mesh

This is not a layer in the classical sense; rather, it forms a hypercube-structured arrangement of multiplicative neurons (as opposed to additive neurons used above) with no activation function. Therefore, we also name this "layer" the Core Mesh. The inputs of the Core Mesh are the output vectors $\mathbf{f}_n^C(x_n)$ of the Complexity Layers of the inputs x_n. The tensor-valued output $\mathcal{F}^{CM}(\mathbf{x}) \in \mathbb{R}^{C^N}$ of the Core Mesh is generated by the tensor product of these inputs as:

$$\begin{aligned} [\mathcal{F}^{CM}(\mathbf{x})]_{c_1,c_2,\ldots c_N} &= \\ = \prod_{n=1}^{N} \left(\sum_{r=1}^{R_n} s_{n,r}(x_n, \mathbf{g}_n) u_{n,r,c_n} \right) &= \prod_{n=1}^{N} f_{n,c_n}^C(x_n). \end{aligned} \tag{10}$$

A two-input case is depicted in Figure 4.

E. Complete Neural Mesh Architecture

We arrive at the complete Neural Mesh architecture, when we add a final Output Layer having connection gains $b_{c_1,c_2,\ldots c_N}$ to the Core Mesh. Therefore each output of the Core Mesh is multiplied by $b_{c_1,c_2,\ldots c_N}$ as

$$[\mathcal{F}^{CM}(\mathbf{x})]_{c_1,c_2,\ldots c_N} b_{c_1,c_2,\ldots c_N} \tag{11}$$

and then summed as

$$f(\mathbf{x}) = \sum_{c_1=1}^{C_1} \ldots \sum_{c_n=1}^{C_N} [\mathcal{F}^{CM}(\mathbf{x})]_{c_1,c_2,\ldots c_N} [\mathcal{B}]_{c_1,c_2,\ldots c_N}. \tag{12}$$

When we substitute the above components we have

$$f(\mathbf{x}) = \sum_{c_1=1}^{C_1} \dots \sum_{c_n=1}^{C_N} \prod_{n=1}^{N} f_{n,c_n}^{C}(x_n)[\mathcal{B}]_{c_1,c_2,\dots c_N} = \quad (13)$$

that is

$$f(\mathbf{x}) = \quad (14)$$

$$\sum_{c_1=1}^{C_1} \dots \sum_{c_n=1}^{C_N} \prod_{n=1}^{N} \left(\sum_{r=1}^{R_n} s_{n,r}(x_n, \mathbf{g}_n) u_{n,r,c_n} \right) [\mathcal{B}]_{c_1,c_2,\dots c_N}$$

which can be written using the tensor-vector product as

$$f(\mathbf{x}) = \mathcal{B} \bar{\boxtimes}_{n=1}^{N} (\mathbf{s}_n(x_n, \mathbf{g}_n)\mathbf{U}_n) \,. \quad (15)$$

A 2D input case is depicted in Figure 5. A 3D input case is depicted in Figure 6.

Consider a case when $\mathbf{f}(\mathbf{x})$ is a vector-valued function. Then each output $f_l(\mathbf{x})$ has its own gains as

$$[\mathbf{f}(\mathbf{x})]_l = \sum_{c_1=1}^{C_1} \dots \sum_{c_n=1}^{C_N} \prod_{n=1}^{N} f_{n,c_n}^{C}(x_n)[\mathcal{B}_l]_{c_1,c_2,\dots c_N} \,. \quad (16)$$

This means we can create an $N+1$ dimensional tensor $\mathcal{B} \in \mathbb{R}^{C^N \times L}$ where the size of the last dimension is L, such that each entry $c_1, c_2, \dots c_N$ is a vector with size L. Therefore the above product becomes

$$\mathbf{f}(\mathbf{x}) = \mathcal{B} \bar{\boxtimes}_{n=1}^{N} (\mathbf{s}_n(x_n, \mathbf{g}_n)\mathbf{U}_n) \,. \quad (17)$$

Consider a case when we have a higher-order structure with a tensor-valued function $\mathcal{F}(\mathbf{x}) \in \mathbb{R}^{J^O}$. Then each entry $c_1, c_2, \dots c_N$ of $\mathcal{B}$ must be a tensor with the size of $J_1 \times \dots \times J_O$. This leads to

$$\mathcal{F}(\mathbf{x}) = \mathcal{B} \bar{\boxtimes}_{n=1}^{N} (\mathbf{s}_n(x_n, \mathbf{g}_n)\mathbf{U}_n) \,, \quad (18)$$

where $\mathcal{B} \in \mathbb{R}^{C^N \times J^O}$. Note that this transfer function of the Neural Mesh is identical to TP model form resulted by the TP model transformation [1]–[10].

IV. Design-Oriented Representation, and Formal Verifiability

The outputs of the Complexity Layer, referred to as the complexity functions, are equivalent to the weighting-function system of the TP model. These functions are also known as antecedent membership functions in fuzzy-system representations and scheduling functions in polytopic system representations. A recent publication [10] highlights their crucial role in control design and optimization.

The TP model transformation provides various tools for manipulating these functions in order to obtain favorable control-theoretic properties. In the Neural Mesh, however, the manipulation of the corresponding weighting functions is reduced to the adjustment of the connection-weight matrices $\mathbf{U}_n$. Consequently, structural modifications that would traditionally be performed through TP model transformation can be interpreted as tuning and training operations within the Neural Mesh architecture.

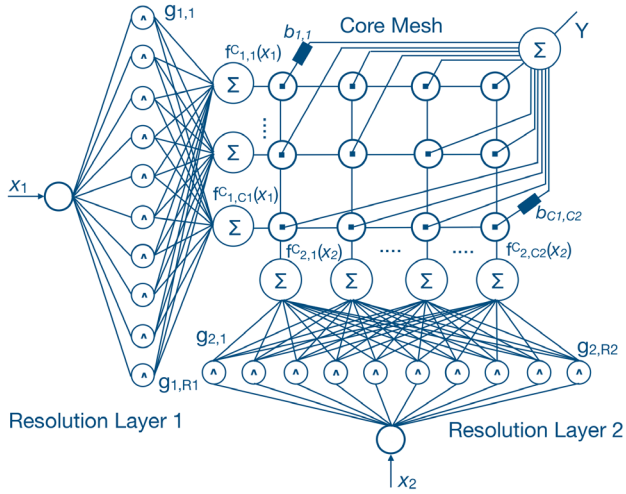

Fig. 5: Two-input Neural Mesh

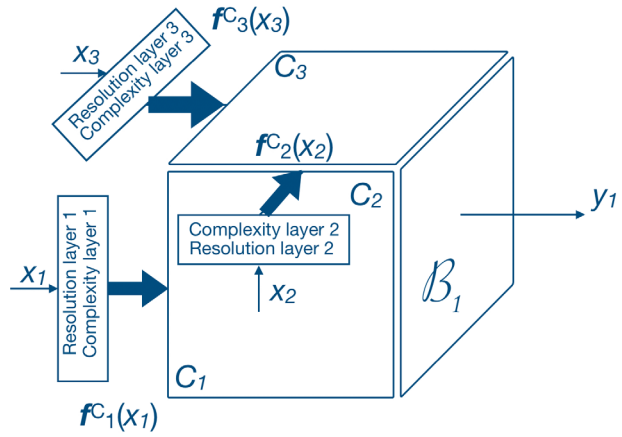

Fig. 6: Three-input, single-output Neural Mesh

Therefore, if further constraints are imposed on the outputs of the Complexity Layer, well-known polytopic formulations widely used in modelling and – for instance – control-theoretic research domains can be obtained. Specifically, if we enforce that the sum of the complexity functions $f_{n,c}^{C}(x_n)$ is always 1 on each dimension and are non-negative – a condition referred to as the Sum-Normalized and Non-Negative (SNNN) condition [1], [33] – as

$$\forall n, x_n : \sum_{c=1}^{C_n} f_{n,c}^{C}(x_n) = 1 \text{ and } \forall n, c, x_n : 0 \leq f_{n,c}^{C}(x_n) \quad (19)$$

then the resulting representation becomes convex, meaning that the output always remains within the convex hull spanned by the vertices $[\mathcal{B}]_{c_1,c_2,\dots c_N}$. Such convex formulations constitute fundamental requirements in several application domains, particularly in control theory and polytopic control design. This also contributes to the boundedness and numerical stability of the Neural Mesh output.

Furthermore, if the CNO (Close-to-Normalized) or IRNO (Inverse-Relaxed-Normalized) [1], [33] condition is imposed

on $\mathbf{f}_n^C(x_n)$, then these functions can naturally express meaningful classifications of the input domain, similarly to how membership functions in fuzzy logic provide a comprehensive representation of a given domain [1]–[3].

We note that the reason why the Neural Mesh uses triangular activation functions in the Resolution Layer, as described in Eq. (3), is that by enforcing these algebraic constraints such as SNNN, CNO via simple matrix operations on U, the model maintains a valid polytopic representation throughout training. This structural guarantee simplifies downstream evaluation steps, such as LMI feasibility checks, although computational scalability remains a consideration for high-dimensional inputs. Imposing the SNNN or CNO constraints on matrix $\mathbf{U}$ – resulting in simple matrix operations – is equivalent to imposing them on the Complexity functions at the output of the Complexity Layer, see Eq. (9). This follows from the closure property of SNNN and CNO matrices under matrix multiplication. Since $\mathbf{s}_n(x_n, \mathbf{g}_n)$ is SNNN or CNO, the product $\mathbf{s}_n(x_n, \mathbf{g}_n)\mathbf{U}_n$ remains SNNN or CNO, respectively, whenever $\mathbf{U}_n$ is chosen to be SNNN or CNO; see [1]–[3]. This has several advantages: (i) convexity is guaranteed by construction at every step of training, meaning the Neural Mesh always represents a valid polytopic model – this enables interleaving learning with control-design procedures such as LMI feasibility evaluation; (ii) the non-negativity and sum-to-one properties of the weights eliminate the vanishing-product problem in the Core Mesh (see Section V-D1); (iii) no nonlinear activation function is needed in the Complexity Layer, thereby preserving the pure HOSVD-based tensor-product structure; and (iv) the constrained rows of $\mathbf{U}$ are directly interpretable as importance distributions over the Complexity Layer cells.

V. Training of the Neural Mesh

All parameters of the Neural Mesh – namely the resolution parameters R_n, the subregions defined by $\mathbf{g}_n$, the weights between the Resolution Layer and Complexity Layer defined by $\mathbf{U}_n$, the number of neurons (rank) in the Complexity Layer defined by C_n, and the weights in the Output Layer defined by the tensor $\mathcal{B}$ – are subject to training. In practice, it is reasonable to start with lower values of the structural parameter R_n in order to capture the overall characteristics of the entire domain, and then to gradually increase R_n to achieve finer resolution. Similar considerations apply to C_n. First, a low-rank Neural Mesh is trained, which essentially captures the dominant or main components of the input-output mapping. Subsequently, the rank of the Neural Mesh can also be increased gradually by introducing additional neurons into the Complexity Layer.

A. Loss Functions

For regression tasks, the raw output $\mathbf{f}(\mathbf{x})$ is used directly and the model is trained with mean squared error (MSE).

For classification tasks with K classes, the output of the Neural Mesh produces K logits and a softmax is applied:

$$\hat{p}_k(\mathbf{x}) = \frac{\exp([\mathbf{f}(\mathbf{x})]_k)}{\sum_{k'=1}^{K} \exp([\mathbf{f}(\mathbf{x})]_{k'})}, \tag{20}$$

and the model is trained with the cross-entropy loss $\mathcal{L} = -\sum_k t_k \log \hat{p}_k$, where t_k is the one-hot target.

B. Optimization

The implementation supports Adam, AdamW, and SGD optimisers. In practice, Adam with a suitable learning rate and mini-batches provides fast convergence. Optional weight decay can be applied for regularization. All parameters of the Neural Mesh – the membership centers (and optionally supports), Complexity Layer weights, and Output Layer weights – are differentiable and trained end-to-end via backpropagation.

C. Setting the Resolution and Complexity

In many cases, the Resolution parameters R_n and the subregions $\mathbf{g}_n$ can be specified independently of the training procedure according to the sensitivity of the individual inputs. Similarly, the number of neurons in the Complexity Layer, defined by C_n, can also be adjusted based on different design considerations.

Increasing the Resolution primarily improves the granularity of the approximated input–output mapping. In contrast, increasing the number of neurons in the Complexity Layer becomes beneficial when further increasing the Resolution no longer improves the approximation performance. In such cases, the representational rank of the Neural Mesh must be increased by introducing additional neurons into the Complexity Layer.

The insertion of additional neurons into the Resolution Layer can be performed in a straightforward manner. Specifically, when a new neuron is inserted between two neighbouring neurons, a new row is inserted into the corresponding matrix $\mathbf{U}_n$ between the rows associated with the neighbouring neurons. The elements of the newly inserted row are determined through interpolation between the two neighbouring rows according to the relative distances of the new neuron from its neighbours.

Consider the case where an additional neuron with offset $g_{n,a}$ is inserted on input n between two neighbouring neurons with offsets $g_{n,r}$ and $g_{n,r+1}$, such that $g_{n,r} < g_{n,a} < g_{n,r+1}$. Then, the weights corresponding to this additional neuron are obtained by linear interpolation between the weights of the two neighbouring neurons.

$$[\mathbf{U}_n]_a = (1-\lambda)[\mathbf{U}_n]_r + \lambda[\mathbf{U}_n]_{r+1}, \tag{21}$$

where

$$\lambda = \frac{g_{n,a} - g_{n,r}}{g_{n,r+1} - g_{n,r}}. \tag{22}$$

This operation does not change the input–output mapping of the Neural Mesh; hence, training can proceed without having to restart from the beginning.

In a similar manner, inserting an additional neuron into the Complexity Layer corresponding to input n is also straightforward. Consider the case where the number of neurons in the Complexity Layer associated with input n is increased from C_n to C_n+1. Then, the tensor $\mathcal{B}$ must be extended with zeros along the corresponding dimension. This operation also does

not change the previously learned input–output mapping of the Neural Mesh.

Once $\mathbf{g}_n$ and C_n are fixed, standard gradient-based training procedures can be applied to further optimize $\mathbf{U}_n$ and $\mathcal{B}$.

D. Further Considerations for Training

Although the network structure and training procedure outlined above is already well-suited to many problem domains, the effectiveness of the training procedure can be further improved using the following methods.

1) Resolution Weight Normalization: The Core Mesh computes the product of per-dimension Complexity Layer activations; when any single factor is zero, the entire product – and its gradient with respect to all other factors – vanishes.

In such cases, it can be helpful to address this "vanishing product" problem by normalizing the weight matrix $\mathbf{U}_n$ prior to the linear transformation in the Complexity Layer. The normalization operates column-wise (per Resolution Layer cell r, across all C_n Complexity Layer cells) and ensures non-negative weights that sum to one. Three modes are available:

- **None**: raw weights are used as-is.
- **Softmax**: $\hat{u}_{c,r} = \exp(u_{c,r}) / \sum_{c'} \exp(u_{c',r})$.
- **Thresholded sum-to-one**: negative weights are clamped to zero and the remaining positive weights are divided by their sum, producing a sparser normalization with exact zeros.

When enabled, this normalization ensures that the Complexity Layer outputs remain non-negative and sum-bounded, which in turn guarantees that all factors in the tensor product of the Core Mesh (see Section III-D) are strictly positive. This eliminates the vanishing-product problem without requiring a nonlinear activation function in the Complexity Layer, thereby preserving the pure linear HOSVD-based tensor-product structure of the TS fuzzy model. Weight normalization in the Complexity Layer is therefore the recommended approach for ensuring stable gradient flow, as it avoids introducing nonlinearities that would obscure the underlying HOSVD-based tensor-product structure.

It should also be noted that this kind of normalization does not change the structure of the model in a mathematical sense, since the gradients will flow back from the loss function at the output through the normalized weights to the raw weights. At inference time, the normalized weights can substitute for the raw weights without any further need for normalization.

2) Parameterizing the Resolution Layer: The activation functions of the Resolution Layer satisfy $\sum_{r=1}^{R_n} s_{n,r}(x_n, \mathbf{g}_n) = 1$ for all x_n. Only the center positions are learnable; they are parameterized as a base value plus cumulative softplus gaps to guarantee strict ascending order:

$$g_{n,r} = g_{n,0} + \sum_{k=1}^{r-1} \left(\text{softplus}(\tilde{\delta}_k) + \epsilon \right), \tag{23}$$

where $\tilde{\delta}_k$ are unconstrained learnable parameters and $\epsilon > 0$ is a small constant. The left and right distances are derived from neighboring centers: $\delta^L_{n,r} = g_{n,r} - g_{n,r-1}$ and $\delta^R_{n,r} = g_{n,r+1} - g_{n,r}$.

3) Optional Activation Function in the Complexity Layer: Although not a necessary component of the architecture, an optional nonlinear activation function σ and learnable bias $\mathbf{b}_n$ may be introduced in the Complexity Layer. When used, for complexity cell c the output becomes:

$$f^C_{n,c}(x_n) = \sigma \left(\sum_{r=1}^{R_n} u_{n,r,c}\, s_{n,r}(x_n, \mathbf{g}_n) + b^{(n)}_c \right). \tag{24}$$

However, when weight normalization (Section V-D1) is enabled, the activation function in the Complexity Layer can be omitted entirely. Experimental results confirm that weight normalization alone yields equivalent performance to using a softplus activation without normalization. The preferred configuration is therefore to use weight normalization without an activation function in the Complexity Layer, as this preserves the direct correspondence with the classical polytopic formulation widely used in modeling and control.

4) Firing Normalization: When firing normalization is enabled, the raw strengths of the Core Mesh are divided by their sum to form a partition of unity:

$$\hat{r}_{c_1,c_2,\ldots c_N} = \frac{[\mathcal{F}^{CM}(\mathbf{x})]_{c_1,c_2,\ldots c_N}}{\sum_{c'_1,c'_2,\ldots c'_N} [\mathcal{F}^{CM}(\mathbf{x})]_{c'_1,c'_2,\ldots c'_N} + \epsilon}. \tag{25}$$

This mirrors the standard TS fuzzy inference normalization step and keeps output magnitudes bounded regardless of the number of input dimensions.

5) Post-Mesh Smoothing: Independently of the above, an optional smoothing function (e.g., softplus) may be applied after the tensor-product summation in the Output Layer. This does not affect the internal tensor-product structure and can improve performance in classification tasks.

VI. CONCLUSION

In conclusion, the Neural Mesh establishes the foundation of a new class of AI architectures referred to as Logical Synthesis Models (LSMs), designed to complement rather than replace Large Language Models. By combining learnability with structured and interpretable representations, the Neural Mesh provides a framework that naturally supports mathematical analysis, verification, synthesis, and control. Inspired by Tensor Product-based modelling and by the functional role of the cerebellum in biological intelligence, the proposed architecture bridges machine learning with engineering modelling, design and control methodologies. This integration opens new opportunities for developing AI systems that not only learn from data but also support transparent reasoning, provable properties, and the systematic design and control of complex physical and abstract systems.

The proposed Neural Mesh is mathematically equivalent to the TP model, which is widely used in system modelling, control design, and controller synthesis. However, from a structural perspective, the Neural Mesh represents the most influential TP-model components — those manipulated by TP model transformation to achieve desirable modelling and control-theoretic properties — as neural connections and trainable parameters. As a result, these components can be

manipulated through neural-network training procedures, enabling functionalities similar to those achieved by TP model transformation already during the system-identification phase.

This capability of the Neural Mesh opens a new research direction that aims to combine advanced neural-network training methodologies with the TP-model properties traditionally determined through TP model transformation. Such an approach may enable the direct learning of Neural Mesh or TP-model structures and characteristics tailored to specific modelling, optimization, and controller-synthesis objectives, reducing the need for subsequent offline TP model transformation procedures.

REFERENCES

[1] P. Baranyi, Y. Yam, and P. Várlaki, *Tensor Product Model Transformation in Polytopic Model Based Control*, ser. Automation and Control Engineering. CRC Press, Taylor & Frances Group, March 2017, ISBN 9781138077782 - CAT K34341.

[2] P. Baranyi, *TP-Model Transformation Based Control Design Frameworks*, ser. Control Engineering. Springer, July 2016, ISBN 978-3-319-19605-3.

[3] P. Baranyi, *Dual-Control-Design: TP and TS Fuzzy Model Transformation Based Control Optimisation and Design*, ser. Topics in Intelligent Engineering and Informatics (TIEI, volume 17). Springer, December 2023.

[4] P. Baranyi, "TP model transformation as a way to LMI-based controller design," *IEEE Transactions on Industrial Electronics*, vol. 51, no. 2, pp. 387–400, 2004. [Online]. Available: DOI: 10.1109/TIE.2003.822037

[5] P. Baranyi, "The generalized TP model transformation for T–S fuzzy model manipulation and generalized stability verification," *IEEE Transactions on Fuzzy Systems*, vol. 22, no. 4, pp. 934–948, 2014. [Online]. Available: DOI: 10.1109/TFUZZ.2013.2278982

[6] P. Baranyi, "Extracting LPV and qLPV structures from state-space functions: A TP model transformation based framework," *IEEE Transactions on Fuzzy Systems*, vol. 28, no. 3, pp. 499–509, 2020. [Online]. Available: DOI: 10.1109/TFUZZ.2019.2908770

[7] P. Baranyi, "Relaxed TS fuzzy model transformation to improve the approximation accuracy/complexity tradeoff and relax the computation complexity," *IEEE Transactions on Fuzzy Systems*, vol. 32, no. 9, pp. 5237–5247, 2024. [Online]. Available: DOI: 10.1109/TFUZZ.2024.3418509

[8] P. Baranyi, "Transition between TS Fuzzy models and the associated convex hulls by TS Fuzzy model transformation," *IEEE Transactions on Fuzzy Systems*, vol. 32, no. 4, pp. 2272–2282, 2024. [Online]. Available: DOI: 10.1109/TFUZZ.2023.3348160

[9] P. Baranyi, "How to vary the input space of a T–S fuzzy model: A TP model transformation-based approach," *IEEE Transactions on Fuzzy Systems*, vol. 30, no. 2, pp. 345–356, 2022. [Online]. Available: DOI: 10.1109/TFUZZ.2020.3038488

[10] P. Baranyi, "On the crucial role of antecedent-consequent structure selection in LMI-based PDC control of TS fuzzy systems," *IEEE Transactions on Fuzzy Systems*, no. accepted, p. in print, 2026. [Online]. Available: DOI: 10.1109/TFUZZ.2026.3678634

[11] K. Tanaka and H. Wang, *Fuzzy control systems design and analysis: a linear matrix inequality approach*. John Wiley and Sons, Inc., U.S.A., 2001.

[12] C. Scherer and S. Weiland, "Linear matrix inequalities in control," *Lecture Notes, Dutch Institute for Systems and Control*, Delft, The Netherlands, 2000. [Online]. Available: http://www.cs.ele.tue.nl/sweiland/lmi.htm

[13] Y. Qui, J. H. Park, Park, C. Hua, and X. Wang, "Stability analysis of time-varying delay t–s fuzzy systems via quadratic-delay- product method," *IEEE Transactions on Fuzzy Systems*, vol. 31, no. 1, pp. 129–137, January 2023. [Online]. Available: DOI: 10.1109/TFUZZ.2022.3141237

[14] A. Ait Ladel, A. Benzaouia, R. Outbib, and M. Ouladsine, "Integrated state/fault estimation and fault-tolerant control design for switched t–s fuzzy systems with sensor and actuator faults," *IEEE Transactions on Fuzzy Systems*, vol. 30, no. 8, pp. 3211–3223, 2022. [Online]. Available: DOI: 10.1109/TFUZZ.2021.3107751

[15] Z. Sheng, C. Lin, B. Chen, and Q.-G. Wang, "Stability and stabilization of t–s fuzzy time-delay systems under sampled-data control via new asymmetric functional method," *IEEE Transactions on Fuzzy Systems*, vol. 31, no. 9, pp. 3197–3209, 2023. [Online]. Available: DOI: 10.1109/TFUZZ.2023.3247030

[16] H.-Y. Sun, H.-G. Han, J. Sun, H.-Y. Yang, and J.-F. Qiao, "Security control of sampled-data t–s fuzzy systems subject to cyberattacks and successive packet losses," *IEEE Transactions on Fuzzy Systems*, vol. 31, no. 4, pp. 1178–1188, 2023. [Online]. Available: DOI: 10.1109/TFUZZ.2022.3197312

[17] Y. Wang, C. Hua, and P. Park, "A generalized reciprocally convex inequality on stability and stabilization for t–s fuzzy systems with time-varying delay," *IEEE Transactions on Fuzzy Systems*, vol. 31, no. 3, pp. 722–733, 2023. [Online]. Available: DOI: 10.1109/TFUZZ.2022.3187180

[18] Y. Ren, D.-W. Ding, Q. Li, and X. Xie, "Static output feedback control for t–s fuzzy systems via a successive convex optimization algorithm," *IEEE Transactions on Fuzzy Systems*, vol. 30, no. 10, pp. 4298–4309, 2022. [Online]. Available: DOI: 10.1109/TFUZZ.2022.3146987

[19] W. Chen, Z. Fei, X. Zhao, and M. V. Basin, "Generic stability criteria for switched nonlinear systems with switching-signal-based lyapunov functions using takagi–sugeno fuzzy model," *IEEE Transactions on Fuzzy Systems*, vol. 30, no. 10, pp. 4239–4248, 2022. [Online]. Available: DOI: 10.1109/TFUZZ.2022.3146975

[20] Y. Kang, L. Yao, and H. Wang, "Fault isolation and fault- tolerant control for takagi–sugeno fuzzy time-varying delay stochastic distribution systems," *IEEE Transactions on Fuzzy Systems*, vol. 30, no. 4, pp. 1185–1195, 2022. [Online]. Available: DOI: 10.1109/TFUZZ.2021.3053320

[21] X.-P. Xie, Q. Wang, M. Chadli, and K. Shi, "Relaxed observer-based state estimation of discrete-time takagi–sugeno fuzzy systems based on an augmented matrix approach," *IEEE Transactions on Fuzzy Systems*, vol. 30, no. 9, pp. 4025–4030, 2022. [Online]. Available: DOI: 10.1109/TFUZZ.2021.3129854

[22] X. Xie, C. Wei, Z. Gu, and K. Shi, "Relaxed resilient fuzzy stabilization of discrete-time takagi–sugeno systems via a higher order time-variant balanced matrix method," *IEEE Transactions on Fuzzy Systems*, vol. 30, no. 11, pp. 5044–5050, 2022. [Online]. Available: DOI: 10.1109/TFUZZ.2022.3145809

[23] X. Xie, F. Yang, L. Wan, J. Xia, and K. Shi, "Enhanced local stabilization of constrained n-ts fuzzy systems with lighter computational burden," *IEEE Transactions on Fuzzy Systems*, vol. 31, no. 3, pp. 1064–1070, 2023. [Online]. Available: DOI: 10.1109/TFUZZ.2022.3187182

[24] J. Song and X.-H. Chang, "Induced L_∞ quantized sampled-data control for t–s fuzzy system with bandwidth constraint," *IEEE Transactions on Fuzzy Systems*, vol. 31, no. 3, pp. 1031–1040, 2023. [Online]. Available: DOI: 10.1109/TFUZZ.2022.3193446

[25] Y. Qiu, C. Hua, and Y. Wang, "Nonfragile sampled-data control of t–s fuzzy systems with time delay," *IEEE Transactions on Fuzzy Systems*, vol. 30, no. 8, pp. 3202–3210, 2022. [Online]. Available: DOI: 10.1109/TFUZZ.2021.3107748

[26] M. S. Aslam, P. Tiwari, H. M. Pandey, and S. S. Band, "Observer-based control for a new stochastic maximum power point tracking for photovoltaic systems with networked control system," *IEEE Transactions on Fuzzy Systems*, vol. 31, no. 6, pp. 1870–1884, 2023. [Online]. Available: DOI: 10.1109/TFUZZ.2022.3215797

[27] Q. Lv and S. Fang, "A stability analysis method for high-speed magnetically suspended rotors based on tensor product model transformation," *IEEE Transactions on Transportation Electrification*, vol. 11, no. 2, pp. 5201–5210, 2025. [Online]. Available: DOI: 10.1109/TTE.2024.3476674

[28] B.-S. Chen, Y.-Y. Tsai, and M.-Y. Lee, "Robust decentralized formation tracking control for stochastic large-scale biped robot team system under external disturbance and communication requirements," *IEEE Transactions on Control of Network Systems*, vol. 8, no. 2, pp. 654–666, 2021. [Online]. Available: DOI: 10.1109/TCNS.2021.3087621

[29] Y. Wang, S. Xu, and C. K. Ahn, "Finite-time composite antidisturbance control for t–s fuzzy nonhomogeneous markovian jump systems via asynchronous disturbance observer," *IEEE Transactions on Fuzzy Systems*, vol. 30, no. 11, pp. 5051–5057, 2022. [Online]. Available: DOI: 10.1109/TFUZZ.2022.3160303

[30] S. Tiko and F. Mesquine, "Constrained control for a class of ts fuzzy systems," *IEEE Transactions on Fuzzy Systems*, vol. 31, no. 1, pp. 348–353, 2023. [Online]. Available: DOI: 10.1109/TFUZZ.2022.3182775

[31] B. Visakamoorthi, P. Muthukumar, and H. Trinh, "Reachable set estimation for t–s fuzzy markov jump systems with time-varying 8 delays via membership function dependent H_∞ performance," *IEEE Transactions on Fuzzy Systems*, vol. 30, no. 11, pp. 4980–4990, 2022. [Online]. Available: DOI: 10.1109/TFUZZ.2022.3164799

[32] J. Hu, X. Sun, M. Zhang, and P. Shi, "An offline fuzzy model-predictive control approach using cache," *IEEE Transactions on Fuzzy Systems*, vol. 30, no. 10, pp. 4504–4514, 2022. [Online]. Available: DOI: 10.1109/TFUZZ.2022.3154436

[33] P. Várkonyi, D. Tikk, P. Korondi, and P. Baranyi, "A new algorithm for RNO-INO type tensor product model representation," in *Proceedings of the 9th IEEE International Conference on Intelligent Engineering Systems*, 2005, pp. 263–266. [Online]. Available: DOI: 10.1109/INES.2005.1555170

Péter Baranyi received the Ph.D. degree in informatics in 1999 and the Doctor of Science (D.Sc.) degree from the Hungarian Academy of Sciences in 2006.
He is the founder and principal developer of the Tensor Product (TP) model transformation frame-work for control design and optimization. He introduced the framework in 19 papers published in IEEE Transactions journals and three scientific monographs. He is the recipient of several international awards, including the Sigma Xi Investigator Award, the Kimura Award, and the International Dénes Gábor Award. He currently leads the AI NEXT project supported by the Hungarian NKFIH HU-RIZONT program. His current research interests include TP model and convex hull manipulation based control optimization.

Ádám B. Csapó obtained his PhD degree at the Budapest University of Technology and Economics in 2014. Since 2023, he has been teaching and conducting research at Corvinus University of Budapest.
Dr. Csapó's earlier research focused on soft computing tools for developing cognitive infocommunication channels in virtual collaboration environments, with the goal of enabling users to communicate with each other and with their spatial surroundings in novel and effective ways. He has also contributed to the development of a commercial desktop VR platform designed for both educational and industrial applications. More recently, his work has centered on exploratory data analysis and model transformation methods that enhance interpretability and explainability. Dr. Csapó has co-authored more than 100 publications, including two books and over 40 journal articles.