Reaction Chemistry & Engineering

ROYAL SOCIETY OF CHEMISTRY

PAPER

View Article Online
View Journal

Cite this: DOI: 10.1039/d5re00290g

Neural tanks-in-series: a physics-guided neural network extension of the tanks-in-series model for enhanced flow reactor and reaction modelling

Sebastian Knoll, *[a] Klara Silber,[bc] Christopher A. Hone, [bc] C. Oliver Kappe, [bc] Martin Steinberger[a] and Martin Horn[a]

Received 4th July 2025,
Accepted 26th August 2025

DOI: 10.1039/d5re00290g

rsc.li/reaction-engineering

This paper introduces the neural tanks-in-series (NTiS) model, an extension of the traditional tanks-in-series (TiS) model using physics-guided neural networks (PGNNs). The NTiS model integrates physical principles with data-driven approaches to improve the accuracy and reliability of flow reactor modeling. The NTiS can optimize physical parameters and learn unmodeled dynamics while ensuring physically feasible predictions, even for out-of-domain predictions. The approach is validated using simulations and experimental data from a Paal–Knorr pyrrole reaction, demonstrating its capability to model flow reactor systems under varying conditions. The NTiS framework offers a new, robust, and flexible tool for advancing chemical flow reactor modeling.

1 Introduction

The chemical industry relies heavily on mathematical modeling and simulation to optimize processes, improve efficiency, and ensure safety.[1–5] In particular, the flow chemistry sector has seen growing interest in automation and advanced modeling, as efficient, scalable, and sustainable chemical flow reactors are crucial for ensuring product quality and meeting economic demands.[6–10]

Thereby, traditional modeling approaches, such as the tanks-in-series (TiS) model, the axial dispersion (AD) model but also computational fluid dynamics (CFD) have provided valuable insights into complex chemical processes.[11] For instance, Sheng *et al.* explored the use of the TiS model to analyze and characterize non-ideal flow patterns. By applying the TiS model, the authors investigated various flow regimes and their impact on the residence time distribution (RTD) of a molten metal.[12] Romero-Gomez *et al.* investigated the AD in a pressurized pipe under various flow conditions. The study focuses on understanding how different flow regimes affect the dispersion of solutes in the pipe. Using experimental data and computational models, the authors analyze the impact of flow velocity, pressure, and pipe geometry on the axial dispersion coefficient.[13]

However, these methods often require extensive computational resources or detailed knowledge of the underlying physical and chemical phenomena. In recent years, machine learning (ML) has emerged as a powerful complement to traditional modeling approaches in process modeling.[14] Beyond neural networks, other surrogate modeling techniques such as Gaussian process regression,[15,16] support vector machines[17,18] and polynomial chaos expansions[19] have been explored to approximate complex process behavior. By leveraging large datasets and advanced algorithms, these data-driven approaches can uncover patterns and relationships not easily captured by conventional models. Such methods hold great promise for accelerating innovation, reducing costs and improving the overall performance of chemical processes.[20] This variety of surrogate models also introduces challenges in model training and generalization, which will be discussed in the following section.

Despite the advantages of ML for advanced chemical process modeling, many surrogate modeling techniques, including neural networks, can face challenges related to data availability. Neural networks in particular often require substantial amounts of data to achieve accurate results. To address this limitation, new methods such as physics-informed neural networks (PINNs) and physics-guided neural networks (PGNNs) have been developed.

In the case of PINNs, known physical relationships such as partial differential equations (PDEs) are incorporated into the neural network training by embedding them directly into the loss function as penalty terms. During training, the network not only minimizes the error between predictions and data but

[a] *Institute of Automation and Control, Graz University of Technology, Inffeldgasse 21b, 8010 Graz, Austria. E-mail: martin.horn@tugraz.at*
[b] *Center for Continuous Synthesis and Processing (CCFLOW), Research Center Pharmaceutical Engineering GmbH (RCPE), Inffeldgasse 13, 8010 Graz, Austria. E-mail: christopher.hone@rcpe.at*
[c] *Institute of Chemistry, University of Graz, NAWI Graz, Heinrichstrasse 28, 8010 Graz, Austria*

also strives to satisfy the PDE constraints. As a result, the model is guided towards more physically consistent and meaningful predictions. This approach enables the use of fewer data compared to traditional data-driven methods while maintaining high prediction accuracy.[21] Moreover, it is possible to maintain parameters of the incorporated physical equations from the training of the PINN.

Thus, PINNs are extensively researched and attract significant interest. For instance, Ngo *et al.* leverage PINNs to incorporate physical laws directly into the training process of a neural network, enhancing the accuracy and reliability of a fixed-bed reactor model for catalytic CO_2 methanation. The paper demonstrates that PINNs can effectively solve the governing equations of the reactor model and identify key parameters with high precision, even with limited data.[22] Moreover, Zheng *et al.* developed a physics-informed recurrent neural network (PIRNN) that combines data and mechanistic models for nonlinear systems. They proved a generalization error bound and integrated it into a Lyapunov-based predictive control tested on a noisy chemical reactor. The hybrid model improved noise rejection and generalization over purely data-driven or physics-based controllers.[23] Trávníková *et al.* present an approach using PINNs for modeling fluid dynamics in stirred tanks. By incorporating the physical laws into the training of the neural network, the models achieve higher accuracy and reliability in predicting flow patterns. The paper demonstrates the effectiveness of this approach through various simulations and experimental validations.[24] Noteworthy is also the work of Alhajeri *et al.*, who addressed overfitting in neural networks for chemical process modeling with noisy data. Using partially connected RNNs, they examined Gaussian and non-Gaussian noise and applied Monte Carlo dropout and co-teaching within a PINN framework. Tested on a large-scale Aspen Plus process, their method improved prediction accuracy and closed-loop control under a Lyapunov-based MPC.[25] Wang and Wu proposed a foundation model for chemical reactor modeling that combines meta-learning with physics-informed fine-tuning. Pretrained on diverse reactor dynamics, it adapts quickly to new processes with minimal data while maintaining physical consistency. The model outperformed conventional approaches across multiple reactor types.[26]

Despite their advantages, PINNs and also traditional ML approaches often struggle with extrapolation beyond their training data. This limitation means that predictions for unseen data cannot always be guaranteed to be feasible or accurate and their out-of-domain behavior is in general not well understood.[27,28]

To overcome this limitation, PGNNs were developed. PGNNs integrate physical knowledge such as governing equations, constraints, or domain specific relationships directly into the neural network architecture and training process. This integration can take several forms, for example by adding extra input features derived from physical variables, by embedding physics based equations or constraints into the loss function, or by designing network structures that reflect known physical laws. In this way, PGNNs guide the learning toward physically consistent solutions, which allows training with less data than traditional neural networks. This approach also improves the ability of the model to generalize and extrapolate to unseen conditions. Due to these advantages, PGNNs are increasingly applied to enhance modeling and prediction in the chemical industry.

For instance, Gallup *et al.* focus on PGNNs to improve hybrid process modeling. Their developed framework simplifies PGNN techniques, speeding training and reducing data needs on reactor simulations. Physics guided loss functions and initialization enhance accuracy and consistency while transfer learning boosts convergence though misuse may hurt generalizability.[29] Alongside, Muralidhar *et al.* presented PhyFlow, a novel physics-guided deep learning framework designed to generate interpretable 3D flow fields. By incorporating physical laws directly into the neural network architecture and emulating the projection method commonly used in CFD simulations, PhyFlow ensures that the generated flow fields comply with fundamental principles such as conservation of mass and momentum. The integration not only enhances the accuracy of the predictions but also provides a level of interpretability that is often lacking in purely data-driven models. The authors demonstrated the effectiveness of PhyFlow through various experiments, showing significant improvements in both accuracy and physical consistency compared to traditional deep learning approaches. The study highlights the potential of PGNNs in advancing the field of fluid dynamics modeling, particularly in scenarios where data is sparse or noisy.[30] Additionally, Panjapornpon *et al.* introduced a PGNN model to enhance the prediction accuracy of acid–base treatments in dynamic tubular reactors. By integrating fundamental physical variables, such as residence time and hydroxide ion concentration, derived from reaction schematics and batch experimental data, the model effectively addressed challenges of high nonlinearity and limited data availability. The resulting PGNN demonstrated the potential for achieving high prediction accuracy.[31] Moreover, Kircher *et al.* embedded physical knowledge into neural ODEs to learn reaction kinetics from integral reactor data. Building on their global reaction neural networks, they improved kinetic model discovery over standard neural ODEs. Applied to industrial reactors, including CO oxidation in hydrogen-rich streams, their method handles stiff systems and uses rich data for accurate kinetics.[32]

While PGNNs have demonstrated significant advancements in the chemical industry, their application to traditional models like the TiS or AD models remains limited. There is research conducted to improve those traditional models like Martin-Dominguez *et al.* who presents an improved version of the traditional TiS model by incorporating additional parameters to better capture the complexities of tracer tests in various flow systems. Besides the number of tanks, they also introduce a dead-space fraction and a bypassing fraction to account for stagnant

zones and zones, where the flow circumvent the intended treatment. By refining the TiS model, the study aims to provide more accurate interpretations of tracer test data.[33] Moreover, Shahin *et al.* extend the traditional TiS model to capture the dynamic behavior of solid oxide fuel cells (SOFC) with direct internal reforming. The study emphasizes the importance of modeling the interactions between reforming processes and electrochemical reactions within the TiS framework. Key findings demonstrate that the extended TiS model effectively predicts the SOFC system's transient responses and can be used to optimize its performance under varying conditions.[34] Dutta *et al.* present a dynamic TiS model to simulate gas–liquid interactions in a trickle bed reactor designed for gas fermentation. Their extension incorporates dynamic behavior by introducing time-dependent variables and parameters to capture transient states. The model accounts for gas holdup, liquid holdup, and mass transfer between phases, providing a more accurate representation of the reactor's performance under varying operational conditions. The enhanced TiS model offers improved predictions for gas–liquid interactions, aiding in the optimization and scale-up of gas fermentation processes.[35]

Despite advancements in the TiS model, current approaches still struggle to accurately capture complex, nonlinear reactor dynamics, particularly under varying operating conditions or when extrapolating beyond training data. Additionally, purely data-driven models often lack physical consistency, limiting their reliability for process optimization and control. Combining PGNNs with the TiS model offers a powerful solution to these challenges. The TiS model is well-studied, well-established, and relatively simple, making it a robust foundation for such an extension. To the best of our knowledge, no PGNN-enhanced TiS models have been developed so far. Therefore, we propose a PGNN-enhanced TiS model that improves prediction accuracy by learning complex reactor behaviors while enforcing physical constraints to ensure physically feasible and robust predictions even in out-of-domain scenarios. This integration addresses key limitations of conventional TiS models and provides a more reliable, generalizable tool for advanced flow reactor modeling.

The paper is structured as follows: first, one can find the theoretical background on the traditional TiS model, the concepts of PGNNs and the training process of neural networks. Next, we introduce our neural tanks-in-series (NTiS) approach, detailing the necessary insights and derivations. This is followed by simulations to validate the proposed approach. Next, the NTiS approach is applied to experimental data in a case study. Herein, we demonstrate the applicability to real-world scenarios for enhancing the modeling and prediction of flow reactor systems. We show that the NTiS approach delivers refined predictions and can extrapolate to unseen data while ensuring that predictions remain within physically feasible bounds.

2 Theoretical background

In this section, one can find the necessary insights and notations of the TiS model, how to train a neural network and the concepts and ideas of a PGNN.

2.1 Tanks-in-series model

The TiS model is a widely used approach developed specifically for tubular flow reactors to describe deviations from ideal plug-flow behavior. The model is chosen as it provides a simple but physically interpretable framework to capture non-ideal flow characteristics. In the model, a non-ideal tubular flow reactor is divided into a series of interconnected, perfectly mixed tanks. Each tank represents a small segment of the flow reactor, and it is assumed that the concentration within each tank is the same. The setup allows for the simulation of gradual changes in concentration, such as the dispersion of a species or the progression of a reaction, while the substance moves through the series of tanks.

In Fig. 1 a schematic overview of the TiS model is shown. As one can see, a flow reactor is split in N perfectly mixed tanks whereby an inflowing species is propagating through each tank. The concentration of the i-th species within the j-th tank is represented by $C_i^j(t)$ whereby t denotes the time.

The change of the i-th concentration within the j-th tank can be described by

$$\frac{\mathrm{d}C_i^j(t)}{\mathrm{d}t} = \frac{q(t)}{\Delta V^j}\left[C_i^{j-1}(t) - C_i^j(t)\right] + r_i^j, \tag{1}$$

where ΔV^j represents the volume of the j-th tank, $q(t)$ represents the volumetric flow rate, and r_i^j accounts for any changes of the i-th concentration within the j-th tank due to reactions. The latter term can thereby depend on the temperature $\vartheta(t)$ and other concentrations within the j-th tank. When neglecting the reaction term r_i^j within eqn (1) and splitting the flow reactor in N equally sized tanks, such that $\Delta V^j = \Delta V,\ \forall j \in \{1, 2, \ldots, N\}$ one can describe the change of the i-th concentration within the j-th tank by a low-pass filter of first order.

In the Laplace domain, the relationship between the concentration of the i-th species in the j-th tank and the $(j-1)$-th tank is characterized by the transfer function

$$H^\tau(s) = \frac{\bar{C}_i^j(s)}{\bar{C}_i^{j-1}(s)} = \frac{\frac{1}{\tau}}{s + \frac{1}{\tau}}. \tag{2}$$

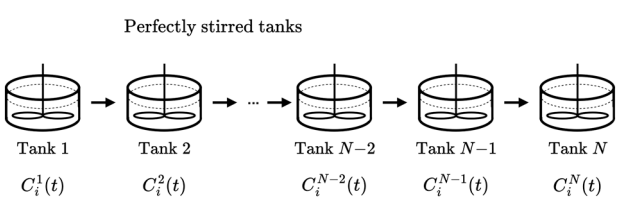

Fig. 1 Schematic overview of the tanks-in-series model.

In the equation, s denotes the Laplace variable, and τ represents the time constant of the low-pass filter for a constant flow rate $q(t) = q$, as defined by

$$\tau = \frac{\Delta V}{q}. \tag{3}$$

For the first tank, the inflowing concentration is the concentration flowing into the reactor system and thus $C_i^0(t) = C_i^{\text{in}}(t)$.

The time constant τ has a significant influence on the behavior of how a species is flowing from one tank to the next. Thus, the time constant τ also affects, how the species is flowing through the entire reactor system. In Fig. 2 the step response of $H^\tau(s)$ for different time constants τ is depicted. The figure represents, how a species is flowing from one tank into the next when the concentration $C_i^{j-1}(t)$ is changed from 0 mol L^{-1} to 0.1 mol L^{-1} at time $t = 0$. This step is denoted with $C_i^{j-1}(t) = 0.1\sigma(t)$ mol L^{-1}.

If the reaction rate r_i^j is not neglected, the model accounts for additional concentration changes caused by chemical reactions within each tank. This introduces nonlinearities and a time-dependent behavior, as reaction rates are influenced by temperature and species concentrations. Incorporating these reaction terms transforms the system into a differential equation that captures both transport and reaction dynamics, as seen in eqn (1).

2.2 Training of a neural network

The training process of a neural network involves several sequential steps to optimize the neural network parameters $\boldsymbol{\theta}$. The goal is, that for a given set of input–output-samples $\{(\mathbf{m}^d, \mathbf{n}^d)$: $d = 1, 2, \ldots, N_s\}$ with N_s samples, the neural network can effectively map all inputs $\mathbf{m}^d$ to the corresponding outputs $\mathbf{n}^d$.

For a fully connected neural network $\mathcal{N}(\mathbf{z},\boldsymbol{\theta})$ with the input $\mathbf{z} = \mathbf{m}^d$, N_L layers, and the form

$$\mathcal{N}(\mathbf{z},\boldsymbol{\theta}) = h^{N_L}(\mathbf{W}^{N_L-1}h^{N_L-1}(\ldots\mathbf{W}^1h^1(\mathbf{W}^0\mathbf{z} + \mathbf{b}^0) + \mathbf{b}^1\ldots) + \mathbf{b}^{N_L-1}), \tag{4}$$

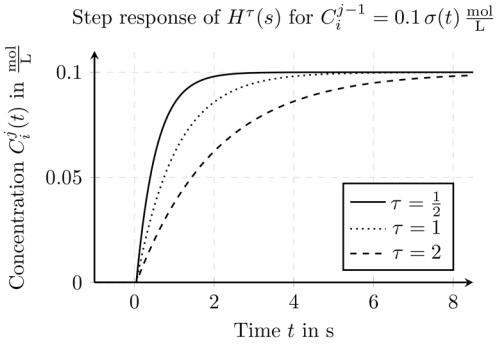

Fig. 2 Step response of the system $H^\tau(s)$ describing the relation of the concentration $C_i^{j-1}(t)$ in the preceding tank $j - 1$ to the concentration $C_i^j(t)$ in tank j for the i-th species.

the network parameters $\boldsymbol{\theta}$ consist of the weights $\mathbf{W}^p$ and bias values $\mathbf{b}^p$ for all $p \in \{0, 1, \ldots, N_L - 1\}$. The weights $\mathbf{W}^p$ represent the connections of the p-th layer to the $(p + 1)$-th layer whereas the vector $\mathbf{b}^p$ represents the bias values within the p-th layer. The term h^p denotes a nonlinear activation function in the p-th layer.

Before one can train the neural network $\mathcal{N}(\mathbf{z},\boldsymbol{\theta})$, the structure of the network such as the number of layers N_L, the activation functions h^p and the number of neurons within each layer must be defined. A neuron is a node in the neural network that processes the input signals from the previous layer, applies the activation function, and produces an output which is forwarded to the next layer. After defining the structure, the parameters of the neural network $\mathcal{N}(\mathbf{z},\boldsymbol{\theta})$ need to be initialized. It is common, to choose random values for the initial weights $\mathbf{W}^p$ and to set the initial bias values $\mathbf{b}^p$ to zero for all $p \in \{0, 1, \ldots, N_L - 1\}$.

After having defined the structure of the neural network $\mathcal{N}$ and having initialized the parameters, one can train the neural network for the given set of input–output-samples. To do so, one can define the prediction loss

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) = &\underbrace{\lambda_p \frac{1}{N_s}\sum_{d=1}^{N_s}\left\|\mathcal{N}\left(\mathbf{m}^d,\boldsymbol{\theta}\right) - \mathbf{n}^d\right\|_2^2}_{\text{penalization}} \\ &+ \underbrace{\lambda_R \sum_{p=0}^{N_L-1}\left(\left\|\mathbf{W}^p\right\|_2^2 + \left\|\mathbf{b}^p\right\|_2^2\right)}_{\text{regularization}},\end{aligned} \tag{5}$$

in which a penalization term and a regularization term is combined. Both terms are weighted relative to one another using the coefficients λ_p and λ_R, respectively. In the equation, $||\cdot||_2$ denotes the L_2-norm and the penalization term accounts for deviations of the predicted output $\tilde{\mathbf{n}}^d = \mathcal{N}(\mathbf{m}^d,\boldsymbol{\theta})$ and the real value $\mathbf{n}^d$ for each sample d and the current network parameters $\boldsymbol{\theta}$. The regularization term mitigates overfitting by penalizing large weights $\mathbf{W}^p$ and bias $\mathbf{b}^p$ values. The loss $\mathcal{L}(\boldsymbol{\theta})$ quantifies how well the network is performing on the current data.

In a so called forward pass the loss $\mathcal{L}(\boldsymbol{\theta})$ for the current network parameters $\boldsymbol{\theta}$ and all the given input–output-samples are calculated. For the optimization of the network parameters $\boldsymbol{\theta}$ one usually makes use of a gradient based optimization such as the Adam optimization algorithm,[36] stochastic gradient descent, or RMSprop.[37] Thus, the gradient of the loss function $\mathcal{L}(\boldsymbol{\theta})$ with respect to each weight $\mathbf{W}^p$ and bias value $\mathbf{b}^p$ has to be determined. To determine those gradients, we perform a backward pass in which we propagate the gradients backward through the network to identify how each parameter contributed to the loss $\mathcal{L}(\boldsymbol{\theta})$. After determining the gradients, one can update the network parameters using

$$\bar{\boldsymbol{\theta}} = \boldsymbol{\theta} - \eta\nabla\mathcal{L}(\boldsymbol{\theta}). \tag{6}$$

Herein, η represents the learning rate, $\boldsymbol{\theta}$ the current network parameters, $\bar{\boldsymbol{\theta}}$ the updated network parameters, and

$\nabla\mathscr{L}(\boldsymbol{\theta})$ the gradient of the loss function with respect to the corresponding network parameters. The learning rate η is a crucial hyperparameter in the optimization process, determining the step size for updating the network parameters $\boldsymbol{\theta}$. A small learning rate η ensures stable convergence but may slow down the training, while a large learning rate η can speed up training but risks overshooting the optimal solution or introducing convergence issues.

In addition to the network parameters $\boldsymbol{\theta}$, which are learned during training, there are hyperparameters, variables that are set prior to training and remain fixed throughout the learning process. Examples of hyperparameters include the number of layers N_L, the activation functions h^p, and the learning rate η. These hyperparameters have a significant impact on training dynamics and model performance. Therefore, while network parameters are optimized through training algorithms, hyperparameters are typically chosen empirically or found through separate tuning procedures or optimization methods.

2.3 Physics-guided neural networks

PGNNs combine physics-based modeling (PBM) with data-driven modeling (DDM), by incorporating physical relationships directly into the neural network architecture and the training process. This integration enhances the prediction accuracy, interpretability, and reduces the amount of data required for training. Additionally, it is possible, to ensure that out-of-domain predictions of the PGNNs remain physically feasible. Predictions of the neural network can be forced to zero ensuring, that only the PBM part of the PGNN remains.

A PGNN can be realized using different structures. In Fig. 3 three common structures of PGNNs are depicted. The structures are very similar as Daw *et al.* described in their paper using PGNN to model the temperature in a lake.[38] In each structure, the input $\boldsymbol{X}$ is used to form a prediction $\tilde{\boldsymbol{Y}}$ of the corresponding real output $\boldsymbol{Y}$. To form this prediction, a neural network $\mathscr{N}$ and a physical model $\mathscr{M}$ is used in combination. In the figure, the first structure (structure a) shows a "hybrid input model". For this structure, the input of the neural network $\mathscr{N}$ is the input $\boldsymbol{X}$ which is extended by the corresponding output of the physical model $\mathscr{M}$. The output of the neural network $\mathscr{N}$ is the predicted overall output $\tilde{\boldsymbol{Y}}$. The second structure (structure b) shows a "residual model" for which the neural network $\mathscr{N}$ predicts only residuals which cannot be covered by the physical model $\mathscr{M}$. This might be due to the fact, that the physical model $\mathscr{M}$ is not accurate enough and not all real-world effects are modeled. Thus the predicted output $\tilde{\boldsymbol{Y}}$ is the sum of the predicted output of the physical model $\mathscr{M}$ and the neural network $\mathscr{N}$. The third structure (structure c) shows a "hybrid input residual model" which is a combination of the two previous structures. The input of the neural network $\mathscr{N}$ is the input $\boldsymbol{X}$ in combination with the predicted output of the physical model $\mathscr{M}$. The neural network $\mathscr{N}$ predicts only residuals, which the physical model $\mathscr{M}$ cannot cover. Thus, the output of the neural network $\mathscr{N}$ is added to the predicted output of a physical model $\mathscr{M}$ to form the final prediction $\tilde{\boldsymbol{Y}}$.

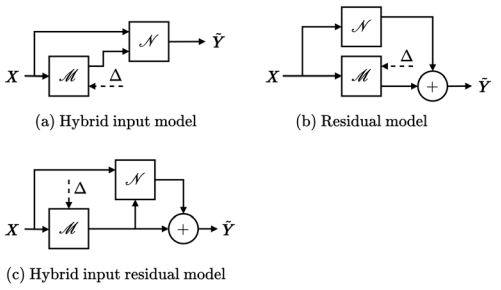

Fig. 3 Common structures of PGNNs.

In all the structures, the physical model $\mathscr{M}$ typically relies on physical parameters, collectively denoted as $\boldsymbol{\Delta}$. Since these parameters may be unknown or only partially accurate, they can be estimated or refined by the PGNN. This refinement process is illustrated in Fig. 3 with a dashed arrow. During training, the PGNN simultaneously trains the neural network and the physical parameters $\boldsymbol{\Delta}$. By incorporating appropriate loss terms and constraints in the training process, it is ensured, that the refined parameters $\boldsymbol{\Delta}$ remain within physically feasible bounds. Consequently, the refined parameters $\boldsymbol{\Delta}$ can be reliably used for out-of-domain predictions, maintaining the physical validity of the outputs coming from the physical model $\mathscr{M}$.

3 Results and discussion

3.1 Neural tanks-in-series model

In this section, we introduce the neural tanks-in-series (NTiS) approach, which extends the traditional TiS model by integrating it into a PGNN framework. The NTiS enhances flow reactor modeling by improving prediction accuracy, and capturing complex dynamics that traditional models cannot represent. Thereby, the NTiS ensures that predictions remain physically feasible, even for out-of-domain predictions where the NTiS has not been explicitly trained.

3.1.1 General structure of the NTiS. The NTiS approach incorporates a "residual model" within each tank. Using this approach, the outputs of the neural network can be easily forced to zero by omitting their contribution to the final output for out-of-domain predictions. Furthermore, the *residual model* is more straightforward to implement in the final structure compared to the "hybrid input residual model". The chosen approach enables the representation of additional dynamics while maintaining the traditional physical flow of the solvent from one tank to the next. Thus, the NTiS concept, illustrated in Fig. 4, replaces each traditional tank with a neural tank (NT). The input to the j-th NT is denoted as $\boldsymbol{X}_k^{j-1}$, and the output as $\boldsymbol{X}_k^{j}$. Here, k denotes the k-th time sample, as the NTiS approach is implemented in a discrete manner. Consequently, the k-th sample corresponds to a point in time $t = kT_d$, where T_d is the

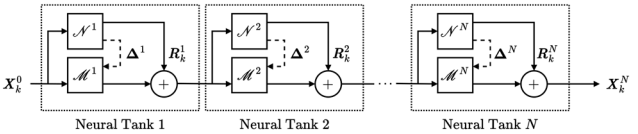

Fig. 4 Schematic overview of the neural tanks-in-series model.

discretization time. In our approach, the input $\boldsymbol{X}_k^{j-1}$ consists of all the concentrations $C_{i,k}^{j-1}$ combined with additional information such as the temperature ϑ_k and flow rate q_k. Similarly, the output $\boldsymbol{X}_k^j$ includes the outflowing concentrations $C_{i,k}^j$ along with the same additional information. The output serves as the input for the subsequent NT $(j+1)$. Each NT j consists of a physical model $\mathcal{M}^j$ and a neural network $\mathcal{N}^j$. The physical model $\mathcal{M}^j$ retains the functionality of the traditional model, accounting for delay effects and reactions within a tank. Meanwhile, the neural network $\mathcal{N}^j$ learns unmodeled dynamics and residuals $\boldsymbol{R}_k^j$, which are combined with the physical model's output before progressing to the next NT. Furthermore, the physical parameters Δ^j for the physical model $\mathcal{M}^j$ are optimized within each NT, enhancing the overall modeling accuracy. To optimize the physical parameters Δ^j, the output layer of the neural network $\mathcal{N}^j$ is extended with additional outputs corresponding to these parameters. Each of these outputs is connected solely to additional bias values, with a linear activation function applied at the final layer. This setup ensures that each additional bias value directly represents a physical parameter. If it is desirable for the physical parameters $\boldsymbol{\Delta}^j$ to vary based on the input, the output vector of the neural network can be extended with full connections to the preceding layers. This allows each NT j to have its own estimation of the physical parameters $\boldsymbol{\Delta}_k^j$ for each time sample k. In any case, the physical parameters are included in the set $\boldsymbol{\theta}$ and are optimized during the training process of the neural network $\mathcal{N}^j$. Consequently, training the PGNN is equivalent to training all the neural networks $\mathcal{N}^j$. In the figure, dashed arrows illustrate this approach, connecting the neural networks $\mathcal{N}^j$ to the physical model $\mathcal{M}^j$.

As discussed in the theoretical background for PGNN (section 2.3), all weights, including the physical parameters, are optimized simultaneously during training. During training, the predictions for the physical parameters Δ^i are constrained by an additional loss term, ensuring that each parameter remains within physically feasible bounds.

In most reactor systems, the physical effects of the reactor remains consistent throughout its entirety. Therefore, we assume that each NT exhibits identical physical effects. Consequently, one can state that $\mathcal{M}^j = \mathcal{M}$ for all $j \in \{1,\ldots, N\}$, and the estimates for the physical parameters Δ^j should be uniform across all NTs j. However, this assumption can generally be relaxed, allowing our NTiS approach to accommodate varying physical parameters within each NT. In combination with the fact, that all neural networks $\mathcal{N}^j$ are trained simultaneously, all neural networks $\mathcal{N}^j$ can be combined into a single global neural network $\mathcal{N}$. This further simplifies the structure and reduces the overall training complexity of the NTiS. The global neural network $\mathcal{N}$ can be designed, to provide a single prediction for the physical parameters $\boldsymbol{\Delta}$ and predictions for the individual residuals $\boldsymbol{R}_k^j$ within each NT j. The predictions for the physical parameters $\boldsymbol{\Delta}$ are shared across all NTs, while the individual residual estimates $\boldsymbol{R}_k^j$ are associated with their respective NTs. These residual estimates are processed through a set of radial basis functions, ensuring that the residuals are forced to zero when predicting out-of-domain data. This approach guarantees that the predictions remain physically feasible, even for unseen data.

In Fig. 5 the refined architecture of the NTiS along with a NT in detail is depicted. To improve the predictions of the neural network $\mathcal{N}$, previous input samples $\left\{\boldsymbol{X}_{k-1}^0, \boldsymbol{X}_{k-2}^0, \ldots, \boldsymbol{X}_{k-N_\mathrm{d}}^0\right\}$, up to a configurable number N_d, are provided as input. To facilitate this, a storage block was introduced before the neural network $\mathcal{N}$. Within the details of a NT, one can see, that the physical model $\mathcal{M}^j$ utilize the predictions of the physical parameters $\boldsymbol{\Delta}$. Before the prediction is passed to the next NT, the corresponding residuals $\boldsymbol{R}_k^j$ are added. The NT depicted in the overview differs slightly from the NT in the schematic shown in Fig. 4, as the neural network is not integrated within the NT. Nevertheless, we will retain this terminology to differentiate it from a traditional tank of the tanks-in-series model. The final predicted output $\tilde{\boldsymbol{Y}}_k$ of the NTiS is the last output $\boldsymbol{X}_k^N$ or parts of it. Consequently, the relation $\tilde{\boldsymbol{Y}}_k = g(\boldsymbol{X}_k^N)$ holds where g represents a function which maps the last output $\boldsymbol{X}_k^N$ to the target structure of the output samples $\boldsymbol{Y}_k$, which may correspond solely to the outflowing concentrations $C_{i,k}^N$ of the final NT, and therefore, of the entire system.

When we assume, that there are P species flowing into the system with a total flow rate q_k of the solute and a temperature ϑ_k, then the input $\boldsymbol{X}_k$ to the NTiS, can be defines as a combination of the inflowing concentrations $C_{i,k}^0 = C_{i,k}^{(\mathrm{in})}$ for all $i \in \{1, 2,\ldots, P\}$, the total flow rate q_k and the temperature ϑ_k. In general, the input to the j-th NT can be written as

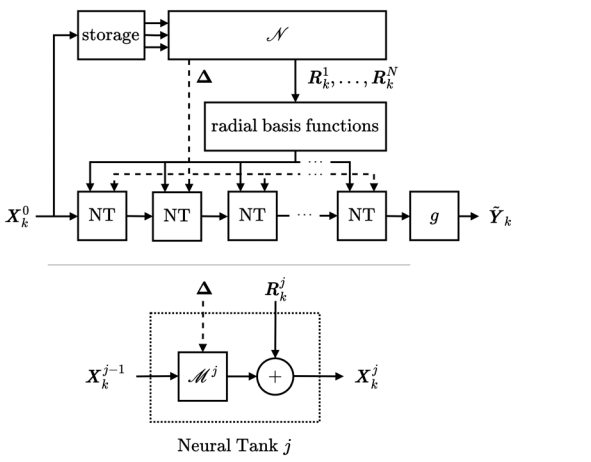

Fig. 5 Overview of the neural tanks-in-series model and an overview of a neural tank.

$$\boldsymbol{X}_k^j = \begin{bmatrix} \boldsymbol{\Pi}_k^j \\ \boldsymbol{\Omega}_k^j \end{bmatrix} | \boldsymbol{\Pi}_k^j = \begin{bmatrix} C_{1,k}^j \\ C_{2,k}^j \\ \vdots \\ C_{P,k}^j \end{bmatrix}, \ \boldsymbol{\Omega}_k^j = \begin{bmatrix} q_k^j \\ \vartheta_k^j \end{bmatrix}, j \in \{0, 1, \ldots, N\}, \quad (7)$$

where $\boldsymbol{\Pi}_k^j$ denotes a vector containing all the concentrations, while $\boldsymbol{\Omega}_k^j$ denotes a vector of all additional quantities, such as the total flow rate q_k^j and the temperature ϑ_k^j. As the total flow rate q_k^j and the temperature ϑ_k^j remain unchanged within a NT j, we theoretically could omit the superscript j. However, since the NTiS could be modified to allow the temperature ϑ_k^j or an additional quantity to vary throughout the flow reactor, we retain the superscript j for the vector $\boldsymbol{\Omega}_k^j$ to ensure generality.

3.2 Physical model $\mathscr{M}^j$

The physical model $\mathscr{M}^j$ within each NT j captures the transport effects from one NT to the next and captures changes due to reactions for each concentration $C_i(t)$. To account for all these effects, one can discretize eqn (1) using the forward Euler method and obtain

$$C_{i,k+1}^j = C_{i,k}^j + T_\mathrm{d} \frac{q_k}{\Delta V} \left[C_{i,k}^{j-1} - C_{i,k}^j \right] + T_\mathrm{d} r_{i,k}^j, \quad (8)$$

where T_d denotes the discretization time, and $\frac{q_k}{\Delta V} = \tau_k$ is representative for the time constant of the low-pass filter. In real-world scenarios, discrepancies may arise between the theoretical time constant τ_k and the effective time constant $\bar{\tau}_k$ due to unmodeled dynamics. To account for such discrepancies, an adjustment factor δ_τ, which will be part of the physical parameters $\boldsymbol{\Delta}$, is introduced such that $\bar{\tau}_k = \delta_\tau \tau_k$.

For the reaction rate $r_{i,k}^j$, in general, any law can be used. One might assume, that the reaction follows the concept of the Arrhenius equation, which relates the reaction rate to temperature and activation energy.[39] The reaction rate forming species i in tank j, while consuming a set of species $\mathscr{R}$, is given by

$$r_i^j(t) = \left[\prod_{n \in \mathscr{R}} C_n^j(t) \right] A_i e^{-\frac{E_i}{R \vartheta_j(t)}}. \quad (9)$$

In the equation, the parameters A_i and E_i are reaction parameters which are in general not precisely known. Thus, the NTiS can refine those parameters by incorporating offsets δ_{A_i} and δ_{E_i} which will also be part of the physical parameters $\boldsymbol{\Delta}$. The discretized reaction rate within the j-th NT can thus be rewritten to

$$r_{i,k}^j = \left[\prod_{n \in \mathscr{R}} C_{n,k}^j \right] (\bar{A}_i + \delta_{A_i}) \ e^{-\frac{\bar{E}_i + \delta_{E_i}}{R \vartheta_k}}. \quad (10)$$

In the equation, $\bar{A}_i$ and $\bar{E}_i$ denote the nominal reaction parameters, R the ideal gas constant and ϑ_k the temperature.

When assuming, that $\boldsymbol{X}_k^{j-1}$ is the input and $\hat{\boldsymbol{X}}_k^j$ the output to the physical model $\mathscr{M}^j$, the model can be defined as

$$\begin{aligned} \mathscr{M}^j: \ \hat{\boldsymbol{\Pi}}_k^j &= \boldsymbol{\Pi}_{k-1}^j + T_\mathrm{d} \delta_\tau \frac{q_{k-1}}{\Delta V} \left[\boldsymbol{\Pi}_{k-1}^{j-1} - \boldsymbol{\Pi}_{k-1}^j \right] + \ldots \\ &\quad + T_\mathrm{d} \xi \left(\boldsymbol{\Pi}_{k-1}^j \right) [\bar{A}_i + \delta_{A_i}] \ e^{-\frac{\bar{E}_i + \delta_{E_i}}{R \vartheta_{k-1}}}. \\ \hat{\boldsymbol{\Omega}}_k^j &= \boldsymbol{\Omega}_k^{j-1}. \end{aligned} \quad (11)$$

Thereby, eqn (8) is applied to each concentration $C_{i,k}^j$ for all $i \in \{1, 2, \ldots, P\}$ and the additional input parameters $\boldsymbol{\Omega}_k^{j-1}$ are forwarded without any changes. In eqn (11), the individual reaction rates for each species are collected in the vector ξ and is thus of size $P \times 1$.

When applying the physical model $\mathscr{M}^j$ to the input $\boldsymbol{X}_k^{j-1}$, we abbreviate it with the notation

$$\hat{\boldsymbol{X}}_k^j = \begin{bmatrix} \hat{\boldsymbol{\Pi}}_k^j \\ \hat{\boldsymbol{\Omega}}_k^j \end{bmatrix} = \mathscr{M}^j \left(\boldsymbol{X}_k^{j-1}, \boldsymbol{\Delta} \right), \quad (12)$$

where $\boldsymbol{\Delta}$ denotes the set of physical parameters and refinement parameters.

3.3 Neural tank model $\mathscr{T}$ and NTiS model $\mathscr{C}$

Overall, one can define a model $\mathscr{T}$ for a NT whereby we apply the physical model $\mathscr{M}^j$ to an input $\boldsymbol{X}_k^{j-1}$ and, before passing the output to the next NT, the corresponding residuals $\boldsymbol{R}_k^j$ predicted by the neural network $\mathscr{N}$ are added. Thus, the NT model can be defined as

$$\begin{aligned} \mathscr{T}: \ \boldsymbol{\Pi}_k^j &= \hat{\boldsymbol{\Pi}}_k^j + \boldsymbol{R}_k^j, \\ \boldsymbol{\Omega}_k^j &= \hat{\boldsymbol{\Omega}}_k^j. \end{aligned} \quad (13)$$

When all residual terms $\boldsymbol{R}_k^j$ are collected in the vector

$$\boldsymbol{\Gamma}_k = \begin{bmatrix} \boldsymbol{R}_k^1 \\ \boldsymbol{R}_k^2 \\ \vdots \\ \boldsymbol{R}_k^N \end{bmatrix}, \quad (14)$$

one can form an abbreviated notation for calculating the output $\boldsymbol{X}_k^j$ of the j-th NT using the physical parameters $\boldsymbol{\Delta}$ and residual terms $\boldsymbol{\Gamma}_k$ as

$$\boldsymbol{X}_k^j = \mathscr{T}(\boldsymbol{X}_k^{j-1}, \boldsymbol{\Delta}, \boldsymbol{\Gamma}_k). \quad (15)$$

Having defined an individual model for a NT, and assuming, that the measured output consists of the measured output concentrations like $\boldsymbol{Y}_k = [C_{1,k}^{(\mathrm{out})} \ C_{2,k}^{(\mathrm{out})} \ \ldots \ C_{P,k}^{(\mathrm{out})}]^T$, one can also define an overall model for the entire series of NTs as

$$\begin{aligned} \mathscr{C}: \ \boldsymbol{X}_k^N &= \underbrace{\mathscr{T}\left(\mathscr{T}\left(\ldots \mathscr{T}\left(\boldsymbol{X}_k^0, \boldsymbol{\Delta}, \boldsymbol{\Gamma}_k \right) \ldots \right) \right)}_{N \text{ times}} = {}^{\circ N}\mathscr{T}\left(\boldsymbol{X}_k^0, \boldsymbol{\Delta}, \boldsymbol{\Gamma}_k \right), \\ \tilde{\boldsymbol{Y}}_k &= g\left(\boldsymbol{X}_k^N \right) = \boldsymbol{\Pi}_k^N. \end{aligned} \quad (16)$$

In the equation, the operator $\circ^N$ represents the N-fold repeated application of the function $\mathscr{T}$. The abbreviated

notation for the repeated application of the NT model can be stated as

$$\tilde{\mathbf{Y}}_k = \mathscr{C}(\boldsymbol{X}_k^0, \boldsymbol{\Delta}, \boldsymbol{\Gamma}_k). \tag{17}$$

3.3.1 Neural network $\mathscr{N}$. The neural network $\mathscr{N}$ of the NTiS can be realized using various architectures, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), or a fully connected neural network. The structure such as the number of hidden layers and neurons per layer can be customized by the user to suit the complexity of the underlying system.

As seen in Fig. 5, for the input $\mathbf{z}_k$ to the neural network $\mathscr{N}$, we combine the current input to the NTiS $\boldsymbol{X}_k^0$ with previous inputs $\boldsymbol{X}_{k-N_\mathrm{d}}^0$ up to a definable number N_d. This approach can increase the prediction accuracy as the neural network $\mathscr{N}$ gains knowledge of previous inputs to the system. The input $\mathbf{z}_k$ to the neural network $\mathscr{N}$ can thus be defined as

$$\mathbf{z}_k = \begin{bmatrix} \boldsymbol{X}_k^0 \\ \boldsymbol{X}_{k-1}^0 \\ \vdots \\ \boldsymbol{X}_{k-N_\mathrm{d}+1}^0 \end{bmatrix}. \tag{18}$$

For the output $\mathbf{y}_k = \mathscr{N}(\mathbf{z}_k, \boldsymbol{\theta})$, we want to have a combination of the physical parameters $\boldsymbol{\Delta}$ and the residual terms $\boldsymbol{\Gamma}_k$. The output of the neural network $\mathscr{N}$ thus read as

$$\mathbf{y}_k = \begin{bmatrix} \boldsymbol{\Delta} \\ \boldsymbol{\Gamma}_k \end{bmatrix}. \tag{19}$$

As described in section 3.1.1, the physical parameters $\boldsymbol{\Delta}$ are obtained by extending the output layer of the neural network $\mathscr{N}$ with an appropriate number of additional bias terms. This approach enables training the neural network $\mathscr{N}$ while simultaneously optimizing the bias terms for the physical parameters $\boldsymbol{\Delta}$.

For the training of the neural network $\mathscr{N}$ we make use of a gradient based optimization. Thereby, a defined loss $\mathscr{L}$ is minimized to optimize all the weights $\mathbf{W}^p$ and bias values $\mathbf{b}^p$ which are collectively denoted in the network parameters $\boldsymbol{\theta}$.

When we assume, that the neural network $\mathscr{N}$ has N_L layers and that each time sample can be used for the training, one can define a cost $\mathscr{L}(\boldsymbol{\theta})$, similar as stated in eqn (5), as

$$\mathscr{L}(\boldsymbol{\theta}) = \lambda_\mathrm{P} \frac{1}{N_s} \sum_{k=1}^{N_s} \left\| \mathscr{C}(\boldsymbol{X}_k^0, \boldsymbol{\Delta}, \boldsymbol{\Gamma}_k) - \boldsymbol{Y}_k \right\|_2^2 + \lambda_\mathrm{R} \sum_{p=0}^{N_\mathrm{L}-1} \left(\|\mathbf{W}^p\|_2^2 + \|\mathbf{b}^p\|_2^2 \right) + \lambda_\mathrm{C} \mathscr{L}_\Delta(\boldsymbol{\theta}). \tag{20}$$

In the cost $\mathscr{L}(\boldsymbol{\theta})$, an additional loss term $\mathscr{L}_\Delta(\boldsymbol{\theta})$ is added to ensure, that the refined parameters $\boldsymbol{\Delta}$ remain within physically feasible bounds. The cost term can be, for example, of a cost term similar to

$$\mathscr{L}_\Delta(\boldsymbol{\theta}) = \sum_\delta e^{(\delta - \delta_\mathrm{min})(\delta - \delta_\mathrm{max})}. \tag{21}$$

Herein, δ represents a physical adjustment parameter out of $\boldsymbol{\Delta}$, δ_min and δ_max the lower and upper bounds of the feasible values for the physical adjustment parameter δ respectively.

For the training process, we have to calculate the gradient of the loss $\mathscr{L}(\boldsymbol{\theta})$ with respect to all the network parameters in $\boldsymbol{\theta}$. For the penalization term one can find, that

$$\frac{\partial \mathscr{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{2\lambda_\mathrm{P}}{N_s} \sum_{k=1}^{N_s} \left\| \mathscr{C}(\boldsymbol{X}_k^0, \boldsymbol{\Delta}, \boldsymbol{\Gamma}_k) - \boldsymbol{Y}_k \right\|_2 \frac{\partial \mathscr{C}(\boldsymbol{X}_k^0, \boldsymbol{\Delta}, \boldsymbol{\Gamma}_k)}{\partial \boldsymbol{\theta}} \tag{22}$$

and thus, that we have to calculate

$$\frac{\partial \mathscr{C}(\boldsymbol{X}_k^0, \boldsymbol{\Delta}, \boldsymbol{\Gamma}_k)}{\partial \boldsymbol{\theta}} = \frac{\partial \mathscr{C}(\boldsymbol{X}_k^0, \boldsymbol{\Delta}, \boldsymbol{\Gamma}_k)}{\partial \boldsymbol{X}_k^0} \frac{\partial \boldsymbol{X}_k^0}{\partial \boldsymbol{\theta}} + \frac{\partial \mathscr{C}(\boldsymbol{X}_k^0, \boldsymbol{\Delta}, \boldsymbol{\Gamma}_k)}{\partial \boldsymbol{\Delta}} \frac{\partial \boldsymbol{\Delta}}{\partial \boldsymbol{\theta}} + \frac{\partial \mathscr{C}(\boldsymbol{X}_k^0, \boldsymbol{\Delta}, \boldsymbol{\Gamma}_k)}{\partial \boldsymbol{\Gamma}_k} \frac{\partial \boldsymbol{\Gamma}_k}{\partial \boldsymbol{\theta}} \tag{23}$$

over all N_s sample points. As $\boldsymbol{X}_k^0$ does not dependent on $\boldsymbol{\theta}$ one can neglect the first part. Moreover, one can see, that $\frac{\partial \boldsymbol{\Delta}}{\partial \boldsymbol{\theta}}$ and $\frac{\partial \boldsymbol{\Gamma}_k}{\partial \boldsymbol{\theta}}$ depend on the structure of the neural network $\mathscr{N}$. The according derivations are calculated within the training process of the neural network itself. All derivations utilized in the implementation are detailed in Appendix A.

3.4 Implementation and simulations

3.4.1 Implementation. The proposed NTiS approach is implemented in Matlab, utilizing the Deep Learning Toolbox.[40] The NTs were designed as custom neural network layers which in Matlab allow users to define forward and backward computations customized for specific applications. This integration enables the NTiS to be represented as a single, unified neural network within Matlab, simplifying both its application and training process. Each custom layer includes the necessary backward path, implemented in alignment with the derived model.

In Fig. 6, the structure of the NTiS implementation in Matlab is presented as a single comprehensive neural network. The neural network $\mathscr{N}$ features a custom parameter layer that predicts the physical parameters $\boldsymbol{\Delta}$. In parallel, a user-defined neural network, such as a fully connected neural network, predicts the residuals $\boldsymbol{\Gamma}_k$. The prediction of the residuals $\boldsymbol{\Gamma}_k$ is processed through a layer of radial basis functions before being forwarded. The input to the NTiS in Matlab includes the current input $\boldsymbol{X}_k^0$ and previous inputs $\boldsymbol{X}_{k-1}^0, \boldsymbol{X}_{k-2}^0, \ldots, \boldsymbol{X}_{k-N_\mathrm{d}}^0$, up to a configurable number N_d. A reduction layer selects only the most recent input $\boldsymbol{X}_k^0$, which is then combined with the physical parameters $\boldsymbol{\Delta}$ and residual predictions $\boldsymbol{\Gamma}_k$ in a concatenation layer. This forms a unified input vector for the subsequent custom NTiS layers, ensuring that each NTiS layer has access to all relevant information while maintaining a single input–output structure. A single input–output structure simplifies implementation in Matlab, as it aligns well with its

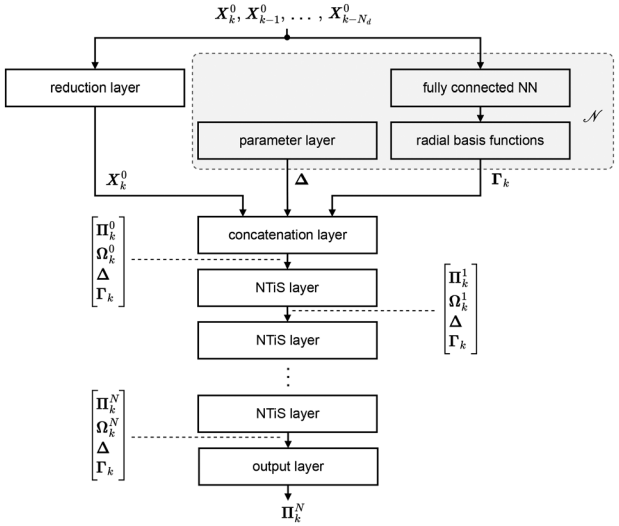

Fig. 6 Schematic of the NTiS implementation within Matlab.

computational framework. Within a custom NTiS layer, the output concentrations $\mathbf{\Pi}_k^j$ are updated based on the input concentrations $\mathbf{\Pi}_k^{j-1}$, additional information $\mathbf{\Omega}_k^j$, physical parameters $\mathbf{\Delta}$, and residual predictions $\mathbf{\Gamma}_k$. Finally, a custom output layer ensures that only the final output concentrations $\mathbf{\Pi}_k^N$ are forwarded.

3.4.2 Simulation with deviating physical parameters. In the first simulation, we highlight that the NTiS can recover underlying physical parameters even from incorrect initial values, thereby validating its basic training functionality. For the simulation we considered a flow reactor where two species flowed into the system. These two species reacted to form a third species within the flow reactor, with the underlying reaction within a tank described by the Arrhenius equation. As system inputs, we used varying concentrations of the first and second species, a varying temperature $\vartheta(t)$, and a varying total flow rate $q(t)$. Given these varying inputs, we simulated the output concentrations using the traditional TiS model, which served as ground truth TiS setup. For all simulations, we used a sample time of T_d = 0.1 seconds, with a total reactor volume of 5 mL evenly distributed across 20 tanks. The traditional TiS model employed fixed reaction parameters A_3 and E_3.

Alongside, we implemented the NTiS model. In contrast to the traditional TiS setup, we intentionally altered the NTiS parameters so that the resulting time constant τ_k and the reaction parameters A_3 and E_3 differed from those of the TiS model used as ground-truth. As a result, the set of physical parameters $\mathbf{\Delta}$ in the NTiS comprises the adjustment factor δ_τ and the offset parameters δ_{A_3} and δ_{E_3}. The latter allow refinement of the pre-exponential factor and activation energy, respectively, while δ_τ accounts for variations in the reactor parameters that cause discrepancies between the theoretical time constant τ_k and the effective time constant $\bar{\tau}_k$.

For the neural network of the NTiS, we used a shallow, fully connected neural network with one hidden layer consisting of 20 neurons. The hyperparameters, such as the number of neurons per layer and the activation function, were determined empirically through manual tuning. In general, it is possible to apply hyperparameter optimization techniques to systematically explore and optimize these structural aspects. The neural network used only the most recent time sample as its input allowing the NTiS model to be set up without the custom reduction layer. For the training of the NTiS, we used the simulated output data from the traditional TiS together with the corresponding inputs. These simulated outputs serve as a stand-in for potential measurements from a real flow reactor, effectively mimicking the actual system response to the given inputs.

During the training process, we showed that the physical adjustment parameters δ_τ, δ_{A_3}, and δ_{E_3} were updated by the neural network such that the resulting time constant τ_k and the reaction parameters A_3 and E_3 closely matched those of the ground truth TiS setup. The adjustment parameter δ_τ and the resulting reaction parameters A_3 and E_3 for the ground truth TiS setup, the initial NTiS model, and the optimized NTiS model can be found in Table 1. The residuals of the NTiS within each NT remained nearly zero, as the predictions were already enhanced by adjusting the physical parameters.

In Fig. 7, the traces for the temperature $\vartheta(t)$, the total flow rate $q(t)$, and the input concentrations $C_1(t)$ and $C_2(t)$ are shown. The input concentration $C_3(t)$ was fixed at zero throughout the experiment. Additionally, the output concentration $C_3(t)$ of species 3, which is formed within the flow reactor, is depicted for three cases: the simulated output of the traditional TiS model and thus representing the ground truth, the initial prediction of the NTiS with parameters containing deviations, and the prediction of the NTiS after the training process. One can see, that the initial prediction of the NTiS shows a significant difference in the shape of the outflowing concentration $C_3(t)$ compared to the actual output. However, after training, the NTiS achieves a highly accurate estimate, demonstrating that its learning process effectively can predict the underlying physical parameters. This trend is also seen, when calculating the mean square error (MSE), denoted as ζ, across all three output predictions for both the initial and optimized predictions of the NTiS. The MSE ζ is defined as

$$\zeta = \frac{1}{3}\frac{1}{N_s}\sum_{i=1}^{3}\sum_{k=0}^{N_s}\left(C_{i,k}^{(\mathrm{out})} - C_{i,k}^{(\mathrm{pred})}\right)^2, \tag{24}$$

where $C_{i,k}^{(\mathrm{out})}$ represents the actual reactor output from the ground truth TiS setup, and $C_{i,k}^{\mathrm{pred}}$ corresponds to the NTiS

Table 1 Adjustment parameter δ_τ, and the resulting reaction parameters A_3 and E_3 for the ground truth TiS setup, the initial and optimized NTiS model

Model	A_3	E_3	δ_τ
Ground truth TiS setup	10.00	15 000	1.20
Optimized parameters of the NTiS model	9.97	14 982	1.19
Initial parameters of the NTiS model	12.00	13 000	1.00

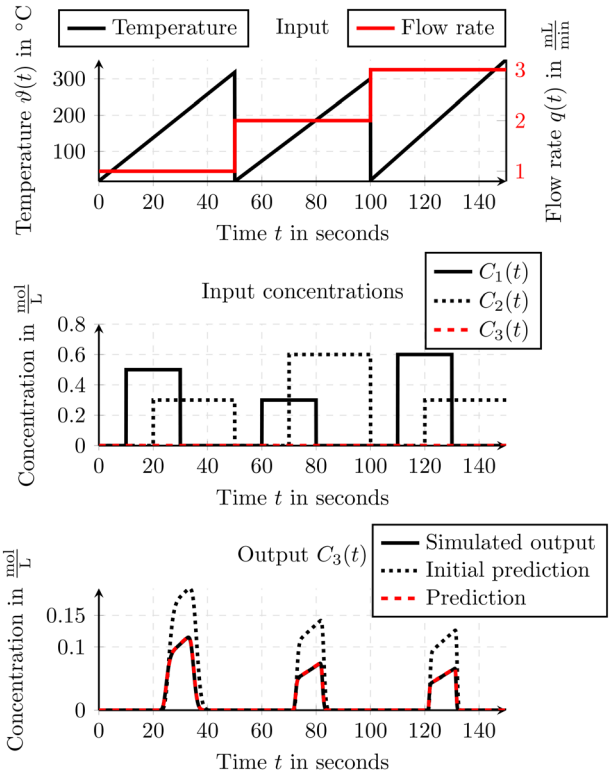

Fig. 7 Reactor simulation showing input variables and output $C_3(t)$ for the traditional tanks-in-series model, the initial prediction of the NTiS, and the prediction of the trained NTiS.

prediction (initial or optimized). The results reveal that the MSE of the initial prediction, $\zeta_{\text{initial}} = 1474 \times 10^{-6}$, is significantly reduced to $\zeta_{\text{Trained}} = 1 \times 10^{-6}$ after training the NTiS.

As seen, in the first simulation, the NTiS mainly focused on adjusting the physical parameters (δ_τ, δ_{A_3}, and δ_{E_3}) while keeping residuals exceptionally small. The minimal residuals confirm that the NTiS does not heavily depend on neural network corrections for parameter deviations. Instead, it achieves high accuracy by aligning its physical parameters with those of the real system which is represented by the traditional TiS model. This highlights the robustness of the NTiS framework, where the neural network component serves as a supplementary mechanism, activating only when necessary. The limited need for residual corrections shows that the NTiS captures the core system dynamics effectively through parameter adjustments, ensuring interpretability and precise predictions.

3.4.3 Simulation with flow-rate-dependent offset. To further evaluate the NTiS training and show its ability to learn unknown physical effects beyond the TiS model, we conducted a second simulation with a flow-rate-dependent offset added to each concentration. Thereby, a concentration offset is introduced to account for additional effects influenced by the flow rate $q(t)$ after each tank. This offset is directly proportional to $q(t)$, meaning its magnitude scales linearly with the flow rate at any given time. The offset might reflect phenomena such as mixing inefficiencies, system disturbances, or chemical reactions occurring at varying flow rates. The result is, that more of species 3 is formed throughout the reactor. Thus, depending on the flow rate $q(t)$, a higher concentration $C_3(t)$ can be obtained at the output. This artificially introduced effect is not accounted for in the traditional TiS model, which therefore cannot address such unmodeled phenomena. The NTiS, on the other hand, can leverage training data to learn these unmodeled effects and improve prediction accuracy. In Fig. 8 one can see the result of the output concentration $C_3(t)$ for the second simulation. For the simulation, the same input concentrations $C_1(t)$, $C_2(t)$ and $C_3(t)$, temperature $\vartheta(t)$, and flow rate $q(t)$ were used as for the first simulation. In the resulting plot the simulated output shows the flow-rate-depending offset. As before, the NTiS was initialized with deviations in the nominal reactor values (time constant τ_k and the reaction parameters A_3 and E_3) resulting in the same physical parameters Δ as for the first simulation. Moreover, as the NTiS is not yet aware of the offsets, the initial prediction using the NTiS shows strong deviations compared to the simulated output. The initial prediction is also similar to what one would expect from the traditional TiS model using the altered parameters, as the flow-rate-dependent offsets are not considered in the traditional TiS model. After training, the NTiS with the same settings as for the first simulation, and using the simulated data for the training, one can see, that the prediction of the NTiS agrees nearly perfectly with the simulated output which shows the mentioned offset depending

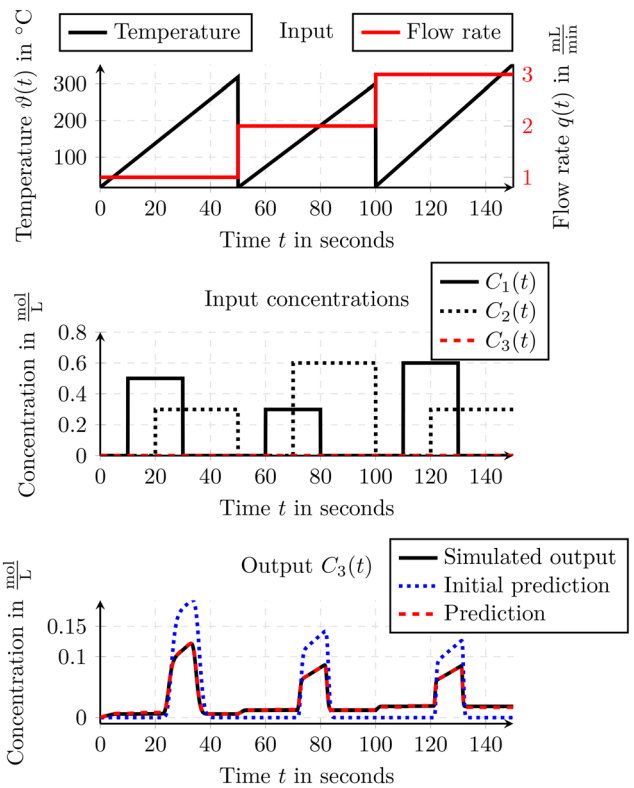

Fig. 8 Reactor simulation with flow-rate-depending offsets. Depicted is the output $C_3(t)$ for the traditional tanks-in-series model, the initial prediction of the NTiS, and prediction of the trained NTiS.

on the flow rate $q(t)$. The reactor parameters are again optimized in such a way, that they match the values used for the simulation to generate the ground truth data.

In Table 2, the adjustment parameter δ_τ and the resulting reaction parameters A_3 and E_3 are reported for the ground truth simulation with a flow-rate-dependent offset, as well as for the initial and optimized NTiS model. The results illustrate how the NTiS training successfully recovers the true physical parameters despite starting from altered initial values. Moreover, as shown in the corresponding figure, the offset dependent on the flow rate $q(t)$ is accurately reconstructed by the NTiS, demonstrating its capability to capture systematic effects not explicitly modeled in the traditional TiS model. This improvement is also evident when comparing the mean squared error (MSE) across all three predictions: the initial NTiS prediction yields $\zeta_{\text{initial}} = 1483 \times 10^{-6}$, which is reduced to $\zeta_{\text{Trained}} = 17 \times 10^{-6}$ after training, highlighting the effectiveness of the parameter adaptation and residual learning in the NTiS approach.

As seen in the second simulation, the NTiS model demonstrated its ability to learn unmodeled dynamics, such as the flow-rate-dependent offset, while still adjusting the physical parameters. The model focused on updating δ_τ, δ_{A_3}, and δ_{E_3}, while also accounting for the flow-rate-dependent offset. The NTiS successfully captured the offset caused by variations in the flow rate $q(t)$. The model's predictions closely matched the simulated output, showing its ability to adjust both physical parameters and unmodeled dynamics. This ability to incorporate both physical parameter updates and the flow-rate-dependent offset highlights the NTiS's robustness, as it can handle complex effects that the traditional TiS model cannot model directly.

Overall, the simulation results clearly demonstrate the effectiveness of the NTiS model in accurately predicting flow reactor outputs by jointly training both physical parameters and neural network weights. This hybrid learning approach enables the NTiS to leverage domain knowledge embedded in the TiS structure while flexibly capturing unmodeled dynamics through residual learning. By balancing updates to the physical parameters with targeted residual estimations, the NTiS can not only recover true underlying process parameters but also adapt to systematic deviations.

3.4.4 Case study and out-of-domain behavior. Within a case study, we apply the NTiS to model a flow reactor system. We show that the NTiS can be used to model the flow reactor system and achieve accurate predictions, even for out-of-domain scenarios. For this purpose, a Paal–Knorr pyrrole reaction, consisting of a single reaction, was utilized. The reaction involves three main components: iso-propanol as the solvent ($C_1(t)$), ethanolamine (NH_2–CH_2–CH_2OH) at a concentration of 1.5 mol L^{-1} ($C_2(t)$), and 2,5-hexanedione at 1.5 mol L^{-1} ($C_3(t)$). These components were introduced into a 5 mL flow reactor at controlled flow rates. Inside the flow reactor, ethanolamine reacts with 2,5-hexanedione to produce the final product, 1-(2-hydroxyethyl)-2,5-dimethylpyrrole ($C_4(t)$), along with two molecules of water. It is assumed, that the underlying temperature dependence of the reaction within a tank can be described by the Arrhenius equation whereby the reaction parameters A_4 and E_4 are not known in detail. The concentrations of both the final product and remaining reactants (2,5-hexanedione) were measured after the flow reactor. The reactor design and the stoichiometry of this single-step Paal–Knorr pyrrole reaction are illustrated in Fig. 9.

In the experiment, we applied different flow rates and temperatures to the flow reactor. The selected temperature $\vartheta(t)$ and total flow rate $q(t)$ profiles, as well as the resulting input concentrations of the reactants (species 2 and 3), are shown in Fig. 10.

During the experiment, the output concentrations of the product and the remaining 2,5-hexanedione were measured and used to train a NTiS model. For the NTiS model we set the number of NTs to 50. Similar as for the simulations, the physical parameters $\boldsymbol{\Delta}$ comprise the adjustment factor δ_τ, as well as the offset parameters δ_{A_3} and δ_{E_3}. The nominal reaction parameters $\bar{A}_4$, $\bar{E}_4$ were selected to lie within a physically meaningful range, and the reactor volume V was adopted from the actual reactor configuration.

For the neural network of the NTiS, we employed a shallow architecture with one hidden layer containing 40 neurons. The network was trained only on the first 4.5 hours of data, while the remaining data beyond this point was kept completely unseen for testing. The hyperparameters of the network and the training process were determined empirically through manual tuning, and no hyperparameter optimization was performed.

Table 2 Reaction parameters δ_τ, A_3 and E_3 for the traditional TiS model, the initial and optimized NTiS model for the second simulation with flow-rate-depending offsets

Model	A_3	E_3	δ_τ
Ground truth simulation (with offsets)	10.00	15 000	1.20
Optimized parameters of the NTiS model	9.89	15 072	1.18
Initial parameters of the NTiS model	12.00	13 000	1.00

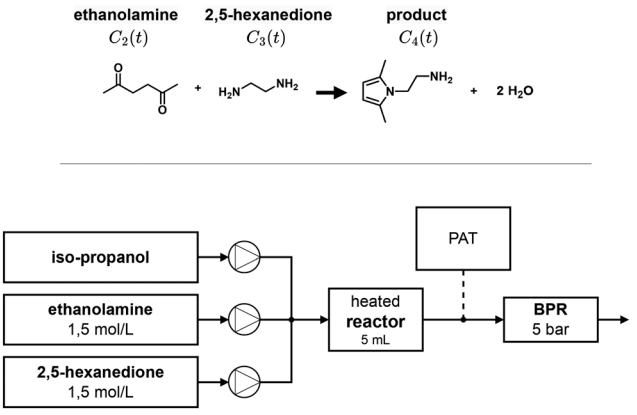

Fig. 9 Reactor setup and stoichiometry of the Paal–Knorr pyrrole reaction.

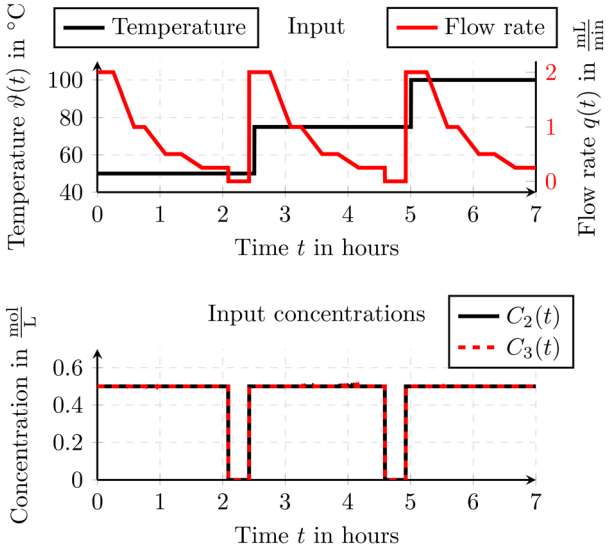

Fig. 10 Inputs to the flow reactor system for the Paal–Knorr pyrrole reaction.

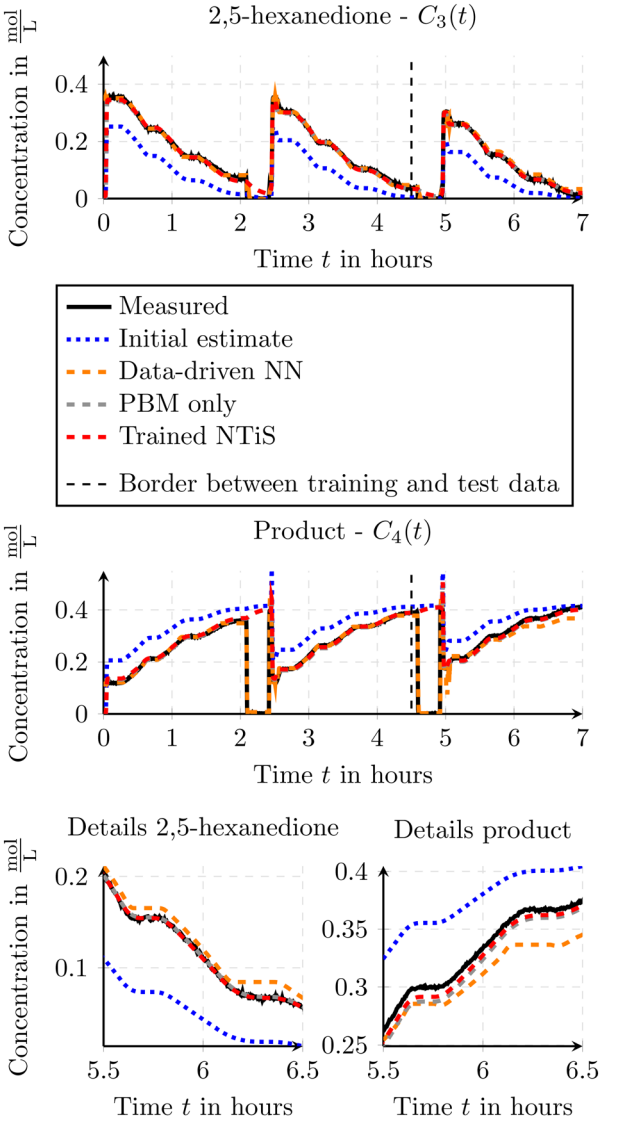

Fig. 11 Predictions for the Paal–Knorr pyrrole reaction. Data to the left of the border was used to train the NTiS and the data-driven neural network while the data to the right are used for test.

Fig. 11 compares the measured output concentrations with the initial NTiS predictions obtained using the initial parameters. During training, the NTiS refined the physical parameters and learned residuals, capturing both modeled and unmodeled effects. Consequently, the trained NTiS predictions shown in the figure demonstrate excellent agreement with the measured data.

Additionally, predictions for out-of-domain data, where the residual contributions of the neural network within the NTiS were set to zero, are included and labeled as "PBM only". Those results are equivalent to a traditional TiS model with optimized physical parameters. The figure shows that while the out-of-domain predictions are an improvement over the initial predictions, the predictions incorporating residuals are slightly more accurate. This indicates that the primary improvements stem from adjustments to the physical parameters, with the residuals contributing only a minor additional boost in accuracy for this experiment.

To further illustrate the behavior in out-of-domain scenarios, we trained a purely data-driven neural network with two hidden layers of 60 neurons each. The neural network was trained exclusively on the first 4.5 hours of data and subsequently applied to estimate the full time series of output concentrations. As shown in the figure, the data-driven neural network achieves excellent agreement within the training domain but exhibits an increasing offset once applied to out-of-domain data. In contrast, both the trained NTiS and the PBM-only predictions maintain physically consistent behavior and do not show such deviations. These results emphasize that while a purely data-driven model can fit in-domain data well, its lack of physical constraints can limit its reliability for extrapolation.

This improvement is further confirmed by the MSE analysis for each individual output shown in Fig. 12. The initial estimates yield the highest MSE, followed by the data-driven neural network and then the PBM-only case. The latter represents out-of-domain predictions or a traditional TiS model with optimized parameters. The trained NTiS consistently achieves the lowest MSE values.

The results of the case study highlight the improved performance of the NTiS compared to the traditional TiS. The NTiS ensures reliable out-of-domain predictions, which purely data-driven methods may not achieve, as illustrated by a neural network that showed deviations on unseen data.

Conclusions

The neural tanks-in-series (NTiS) model presented in this work represents a significant advancement over traditional TiS models through its integration of PGNNs. By combining mechanistic modeling with data-driven approaches, NTiS

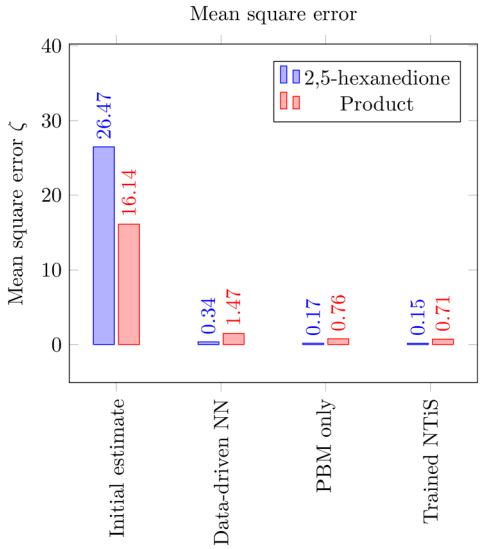

Fig. 12 Mean square error ζ for different models and outputs.

improves prediction accuracy, captures unmodeled dynamics, and ensures physically feasible predictions even in out-of-domain scenarios. It outperforms purely mechanistic TiS models by adapting to varying operating conditions while preserving the physical structure of a flow reactor. Compared to purely data-driven models, NTiS achieves comparable accuracy without the risk of physically infeasible extrapolations. These results highlight the value of NTiS in combining robustness and interpretability with the flexibility of data-driven learning.

Through simulations, we demonstrated that NTiS can refine physical parameters and learn residuals, enabling accurate modeling of complex flow reactor systems. The case study on the Paal–Knorr pyrrole reaction further confirmed its capability to predict real-world reactor behavior under varying conditions. NTiS not only enhances prediction accuracy but also provides reliable estimates when extrapolating beyond the training data.

It should be noted, however, that NTiS relies on the TiS structure for modeling flow reactors. If the assumed topology does not match the actual system, parameter identification may be affected, as the neural components may compensate for structural mismatches. Nevertheless, the NTiS concept can be extended to other reactor types by adapting the underlying physical model, creating opportunities for future studies.

Overall, NTiS provides a robust and flexible framework for enhancing flow reactor modeling. By merging mechanistic understanding with data-driven flexibility, it delivers accurate and physically consistent predictions under varying conditions and out-of-domain scenarios. This makes it particularly suited for flow chemistry applications, digital twins, and PAT-integrated process automation, supporting improved process optimization, real-time decision-making, and accelerated design and scale-up of chemical processes. Its combination of reliability, interpretability, and adaptability underscores its potential for broad impact in both research and industrial settings.

Author contributions

Sebastian Knoll: conceptualization, formal analysis, methodology, validation, investigation, writing original draft, and visualization. Klara Silber: validation, investigation, and conducting experiments. Christopher A. Hone: validation, investigation, supervision, and conducting experiments. C. Oliver Kappe: resources, and supervision. Martin Steinberger: conceptualization, methodology, supervision. Martin Horn: resources, and supervision.

Conflicts of interest

There are no conflicts to declare.

Data availability

Supplementary information: Experimental, hardware and software details and spectra. See DOI: **https://doi.org/10.1039/D5RE00290G**.

The entire implementation for this study, including the MATLAB implementation, experimental data, source code, example scripts, and documentation, is available from Zenodo. The repository data includes: the full MATLAB implementation of the NTiS concept with examples and the case study, example scripts for reproducing the results in this paper, and comprehensive documentation and user guides for the implementation and examples. See DOI: **https://doi.org/10.5281/zenodo.15730195**. Additionally, the MATLAB implementation is accessible on GitHub: **https://github.com/SKenb/NeuralTanksInSeries/**.

Appendices

Appendix A: Gradient computation/backward pass

Appendix A provides a detailed explanation of the backward path computations for a NT realized as custom neural network layer. The backward path is derived by applying the chain rule to compute the gradients of the loss function with respect to the layer's inputs, weights, and biases.

To ensure generality, we consider the possibility of time-varying physical parameters Δ_k^j and allow each NT j to have its own unique set of parameters. To capture this variability, we use the subscript k to indicate time dependency and the superscript j to denote the specific NT.

Fig. 13 illustrates the generalized structure of an NT, whereby the inputs include $\boldsymbol{X}_k^{j-1}$, Δ_k^j, and $\boldsymbol{R}_k^j$, while the output is $\boldsymbol{X}_k^j$. As an NT does not contain internal weights and biases, and when following the derivatives stated in eqn (23), one can find, that the focus is solely on determining

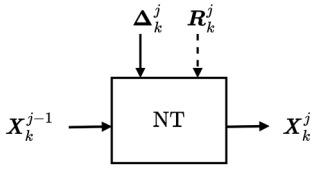

Fig. 13 Generalized inputs and outputs of a NT.

$$\frac{\partial X_k^j}{\partial X_k^{j-1}},\ \frac{\partial X_k^j}{\partial \Delta_k^j},\ \text{and}\ \frac{\partial X_k^j}{\partial R_k^{j-1}}. \tag{25}$$

Derivative with respect to the input X_k^{j-1}. For the derivative with respect to the input X_k^{j-1} one can find

$$\frac{\partial X_k^j}{\partial X_k^{j-1}} = \begin{bmatrix} \frac{\partial \Pi_k^j}{\partial \Pi_k^{j-1}} & \frac{\partial \Pi_k^j}{\partial \Omega_k^{j-1}} \\ \frac{\partial \Omega_k^j}{\partial \Pi_k^{j-1}} & \frac{\partial \Omega_k^j}{\partial \Omega_k^{j-1}} \end{bmatrix} \tag{26}$$

Thereby,

$$\frac{\partial \Omega_k^j}{\partial \Omega_k^{j-1}} = \frac{\partial \hat{\Omega}_k^j}{\partial \Omega_k^{j-1}} = \frac{\partial \Omega_k^{j-1}}{\partial \Omega_k^{j-1}} = \boldsymbol{I}, \tag{27}$$

where $\boldsymbol{I}$ represents the identity matrix of appropriate size, and

$$\frac{\partial \Omega_k^j}{\partial \Pi_k^{j-1}} = 0, \tag{28}$$

as Ω_k^j is not affected by Π_k^{j-1}.

As the output Π_k^j of a NT depends only the previous input (X_{k-1}^{j-1}) and not on the current input X_k^{j-1}, the derivatives $\frac{\partial \Pi_k^j}{\partial \Pi_k^{j-1}}$ and $\frac{\partial \Pi_k^j}{\partial \Omega_k^{j-1}}$ will be zero. This is because the current input X_k^{j-1} does not directly influence Π_k^j. Finally, one can summarize that

$$\frac{\partial X_k^j}{\partial X_k^{j-1}} = \begin{bmatrix} 0 & 0 \\ 0 & \boldsymbol{I} \end{bmatrix}. \tag{29}$$

Derivative with respect to the physical parameters Δ_k^j. Assuming that all the NTs consist of a physical model $\mathcal{M}^j$ with one reaction forming species i, the physical parameters Δ_k^j can be defined as the combination of the delay adjustment $\delta_{\tau,k}^j$, the offset for the pre-exponential factor $\delta_{A_i,k}^j$ and the offset for the activation energy $\delta_{E_i,k}^j$ like

$$\Delta_k^j = \begin{bmatrix} \delta_{\tau,k}^j & \delta_{A_i,k}^j & \delta_{E_i,k}^j \end{bmatrix}^T. \tag{30}$$

This is no general limitation, as the physical model $\mathcal{M}^j$ can include as many reactions as required. More reactions would require to extend the physical parameter vector Δ accordingly.

For the derivative with respect to the physical parameters Δ_k^j, one can find

$$\frac{\partial X_k^j}{\partial \Delta_k^j} = \begin{bmatrix} \frac{\partial \Pi_k^j}{\partial \Delta_k^j} \\ \frac{\partial \Omega_k^j}{\partial \Delta_k^j} \end{bmatrix} = \begin{bmatrix} \frac{\partial \hat{\Pi}_k^j}{\partial \Delta_k^j} + \frac{\partial R_k^j}{\partial \Delta_k^j} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{\partial \hat{\Pi}_k^j}{\partial \Delta_k^j} + 0 \\ 0 \end{bmatrix}, \tag{31}$$

where

$$\frac{\partial \hat{\Pi}_k^j}{\partial \Delta_k^j} = \begin{bmatrix} \frac{\partial \hat{\Pi}_k^j}{\partial \delta_{\tau,k}^j} & \frac{\partial \hat{\Pi}_k^j}{\partial \delta_{A_i,k}^j} & \frac{\partial \hat{\Pi}_k^j}{\partial \delta_{E_i,k}^j} \end{bmatrix}. \tag{32}$$

For the three derivations, one can find

$$\begin{aligned} \frac{\partial \hat{\Pi}_k^j}{\partial \delta_{\tau,k}^j} &= T_\mathrm{d} \frac{q_{k-1}^{j-1}}{\Delta V} \left[\Pi_{k-1}^{j-1} - \Pi_{k-1}^j \right], \\ \frac{\partial \hat{\Pi}_k^j}{\partial \delta_{A_i,k}^j} &= T_\mathrm{d} \xi \left(\Pi_{k-1}^j \right) e^{-\frac{E_i + \delta_{E_i}^j}{R \vartheta_{k-1}^{j-1}}}, \text{and} \\ \frac{\partial \hat{\Pi}_k^j}{\partial \delta_{E_i,k}^j} &= -\frac{T_\mathrm{d}}{R \vartheta_{k-1}^{j-1}} \xi \left(\Pi_{k-1}^j \right) \left[\bar{A} + \delta_{A_i}^j \right] e^{-\frac{E_i + \delta_{E_i}^j}{R \vartheta_{k-1}^{j-1}}}. \end{aligned} \tag{33}$$

When the input Δ_k^j is identical across all time steps k and tanks j, the gradient calculation for each tank and time step, $\frac{\partial \hat{\Pi}_k^j}{\partial \Delta_k^j}$, remains unchanged.

Whenever the input $\Delta_k^j = \Delta$ is shared across all NTs, the overall gradient with respect to Δ is aggregated as the mean of all individual gradients: $\frac{\partial \hat{\Pi}_k^j}{\partial \Delta} = \frac{1}{N} \sum_{k,j} \frac{\partial \hat{\Pi}_k^j}{\partial \Delta_k^j}$, where N is the total number of tanks and time steps. This averaging ensures the collective contribution from all tanks and time steps is accounted for consistently, enabling balanced updates during optimization.

Derivative with respect to the residuals R_k^j. For the derivative with respect to the residuals R_k^j one can find, that

$$\frac{\partial X_k^j}{\partial R_k^j} = \begin{bmatrix} \frac{\partial \Pi_k^j}{\partial R_k^j} \\ \frac{\partial \Omega_k^j}{\partial R_k^j} \end{bmatrix} = \begin{bmatrix} \frac{\partial \hat{\Pi}_k^j}{\partial R_k^j} + \frac{\partial R_k^j}{\partial R_k^j} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 + \boldsymbol{I} \\ 0 \end{bmatrix}. \tag{34}$$

Acknowledgements

The authors gratefully acknowledge funding by Land Steiermark/Zukunftsfonds Steiermark (No. 9003) for acquiring the infrastructure used in this work. The Research Center Pharmaceutical Engineering (RCPE) is funded within the framework of COMET – Competence Centers for Excellent Technologies by BMIMI, BMWET, Land Steiermark and SFG.

The COMET program is managed by the FFG. The laboratory work was funded through the Austrian Research Promotion Agency (FFG) as part of the "Twin4Pharma" project within the COMET Module program.

References

1 G. B. Libotte, F. S. Lobato, F. D. Moura Neto and G. M. Platt, *Ind. Eng. Chem. Res.*, 2022, **61**(9), 3483–3501.
2 A. Rasmuson, B. Andersson, L. Olsson and R. Andersson, *Mathematical Modeling in Chemical Engineering*, Cambridge University Press, 2014.
3 S. Knoll, M. Steinberger, L. Kuchler, A. Azimi, M. Tranninger, S. Sacher and M. Horn, *J. Pharm. Innov.*, 2025, **20**, 72–85.
4 S. R. Upreti, *Process Modeling and Simulation for Chemical Engineers: Theory and Practice*, Wiley, 2017.
5 C. Boyadjiev, *Theoretical Chemical Engineering: Modeling and Simulation*, Springer Berlin Heidelberg, 2010.
6 M. B. Plutschack, B. Pieber, K. Gilmore and P. H. Seeberger, *Chem. Rev.*, 2017, **117**, 11796–11893.
7 C. P. Breen, A. M. Nambiar, T. F. Jamison and K. F. Jensen, *Trends Chem.*, 2021, **3**, 373–386.
8 G. Tom, S. P. Schmid, S. G. Baird, Y. Cao, K. Darvish, H. Hao, S. Lo, S. Pablo-García, E. M. Rajaonson, M. Skreta, N. Yoshikawa, S. Corapi, G. D. Akkoc, F. Strieth-Kalthoff, M. Seifrid and A. Aspuru-Guzik, *Chem. Rev.*, 2024, **124**, 9633–9732.
9 S. Knoll, C. E. Jusner, P. Sagmeister, J. D. Williams, C. A. Hone, M. Horn and C. O. Kappe, *React. Chem. Eng.*, 2022, **7**, 2375–2384.
10 H. Hysmith, E. Foadian, S. P. Padhy, S. V. Kalinin, R. G. Moore, O. S. Ovchinnikova and M. Ahmadi, *Digital Discovery*, 2024, **3**, 621–636.
11 N. Padoin, T. Matiazzo, H. G. Riella and C. Soares, *J. Flow Chem.*, 2024, **14**, 239–256.
12 D.-Y. Sheng and Z. Zou, *Metals*, 2021, **11**, 208.
13 P. Romero-Gomez, Z. Li, C. Y. Choi, S. G. Buchberger, K. E. Lansey and V. T. Tzatchkov, *Water Distribution Systems Analysis 2008*, 2009, pp. 1–10.
14 R. Shen and W. Su, *Pharm. Fronts*, 2023, **05**, e219–e226.
15 C. Lyu, X. Liu and L. Mihaylova, Review of Recent Advances in Gaussian Process Regression Methods, *arXiv*, 2024, preprint, arXiv:2409.08112, DOI: **10.48550/ARXIV.2409.08112**.
16 V. L. Deringer, A. P. Bartók, N. Bernstein, D. M. Wilkins, M. Ceriotti and G. Csányi, *Chem. Rev.*, 2021, **121**, 10073–10141.
17 R. Rodríguez-Pérez and J. Bajorath, *J. Comput.-Aided Mol. Des.*, 2022, **36**, 355–362.
18 S. S. Hossain, B. V. Ayodele, S. S. Ali, C. K. Cheng and S. I. Mustapa, *Sustainability*, 2022, **14**, 7245.
19 J. N. Mueller, K. Sargsyan, C. J. Daniels and H. N. Najm, *SIAM-ASA J. Uncertain. Quantif.*, 2025, **13**, 1–29.
20 S. Du, Z. Huang, L. Jin and X. Wan, *Algorithms*, 2024, **17**, 569.
21 S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi and F. Piccialli, *J. Sci. Comput.*, 2022, **92**, 1–62.
22 S. I. Ngo and Y.-I. Lim, *Catalysts*, 2021, **11**, 1304.
23 Y. Zheng, C. Hu, X. Wang and Z. Wu, *J. Process Control*, 2023, **128**, 103005.
24 V. Trávníková, D. Wolff, N. Dirkes, S. Elgeti, E. von Lieres and M. Behr, *Adv. Comput. Sci. Eng.*, 2024, **2**, 91–129.
25 M. S. Alhajeri, F. Abdullah, Z. Wu and P. D. Christofides, *Chem. Eng. Res. Des.*, 2022, **186**, 34–49.
26 Z. Wang and Z. Wu, *Chem. Eng. Res. Des.*, 2025, **218**, 839–853.
27 L. Fesser, L. D'Amico-Wong and R. Qiu, Understanding and Mitigating Extrapolation Failures in Physics-Informed Neural Networks, *arXiv*, 2023, preprint, arXiv:2306.09478, DOI: **10.48550/arXiv.2306.09478**.
28 A. Bonfanti, R. Santana, M. Ellero and B. Gholami, *Neural Computing and Applications*, 2024, **36**(36), 22677–22696.
29 E. Gallup, T. Gallup and K. Powell, *Comput. Chem. Eng.*, 2023, **170**, 108111.
30 N. Muralidhar, J. Bu, Z. Cao, N. Raj, N. Ramakrishnan, D. Tafti and A. Karpatne, *2021 IEEE International Conference on Data Mining (ICDM)*, 2021, pp. 1246–1251.
31 C. Panjapornpon, P. Chinchalongporn, S. Bardeeniz, K. Jitapunkul, M. A. Hussain and T. Satjeenphong, *Eng. Appl. Artif. Intell.*, 2024, **138**, 109500.
32 T. Kircher, F. A. Döppel and M. Votsmeier, in *34th European Symposium on Computer Aided Process Engineering/15th International Symposium on Process Systems Engineering*, Elsevier, 2024, vol. 53, pp. 817–822.
33 A. Martin-Dominguez, V. G. Tzatchkov, I. R. Martin-Dominguez and D. F. Lawler, *J. Water Supply: Res. Technol.–AQUA*, 2005, **54**, 435–448.
34 S. Hosseini, P. Vijay, K. Ahmed, M. O. Tadé, V. Pareek and R. Utikar, *Int. J. Energy Res.*, 2017, **41**, 1563–1578.
35 S. Dutta, M. Krikeli, H. N. Gavala and I. V. Skiadas, *J. Cleaner Prod.*, 2024, **472**, 143433.
36 D. Kingma and J. Ba, *International Conference on Learning Representations*, 2014.
37 S. Ruder, An overview of gradient descent optimization algorithms, *arXiv*, 2016, preprint, arXiv:1609.04747, DOI: **10.48550/arXiv.1609.04747**.
38 A. Daw, A. Karpatne, W. Watkins, J. Read and V. Kumar, Physics-guided Neural Networks (PGNN): An Application in Lake Temperature Modeling, *arXiv*, 2017, preprint, arXiv:1710.11431, DOI: **10.48550/arXiv.1710.11431**.
39 S. Arrhenius, *Z. Phys. Chem.*, 1889, **4U**, 226–248.
40 The MathWorks, Inc., *Deep Learning Toolbox*, 2025, **https://de.mathworks.com/products/deep-learning.html**, Accessed: 2025-04-11.