# In-Context Learning for System Identification: Recent Advances

Matteo Rufolo [a], Dario Piga [a], Filippo Pura [a], Marco Forgione [a]

[a] *SUPSI, IDSIA - Dalle Molle Institute for Artificial Intelligence, Lugano, Switzerland*

**Abstract**

Traditional system identification typically involves two steps: initially, training data is processed to estimate a model of the system of interest; then, the model is used to predict outputs for new input queries. This paper discusses a recently introduced novel approach to system identification, which is based on an in-context learning paradigm that allows for direct mapping from training data and query inputs to the corresponding system's outputs in a zero-shot fashion, thereby eliminating the need for explicit model estimation. This method relies on a meta model, trained offline to map training data and query inputs into output predictions using a vast set of synthetically generated data. In other words, the meta model is trained to behave like a learned system identification algorithm. This paper presents advancements in this meta learning framework for system identification, including tailored modifications to handle long training sequences and a new formulation for estimating the probability distribution of outputs. Furthermore, the application of a pre-trained meta model for data augmentation in traditional system identification is explored. Various case studies, including nonlinear system identification benchmarks with real-world data, demonstrate the robustness and effectiveness of the meta learning approach.

*Key words:* Meta learning; Artificial neural networks; System identification; Synthetic data.

## 1 Introduction

### 1.1 System identification and meta learning

Given a set of data generated by a dynamic system, classic system identification aims to estimate a model to describe the system's behavior. This requires an expert to select a suitable model structure (e.g., linear *vs* nonlinear, time-invariant *vs* time-varying, state-space *vs* input/output representation, etc.) and an identification approach (e.g., prediction error methods, subspace methods, kernel-based approaches, etc.). Additionally, experts must choose hyper-parameters in numerical optimization approaches often adopted to solve the formulated identification problem (e.g., learning rate in gradient-based methods, regularization weights and stopping criteria to prevent overfitting, etc.). Furthermore, a new model is estimated from scratch for each system of interest, typically without leveraging knowledge previously acquired while identifying similar systems.

Building on the seminal work by some of the authors in [12], this paper presents a novel system identification paradigm inspired by meta learning concepts and techniques [14]. The presented approach differs from classic system identification in several ways:

- A meta model, representing a class of related systems, is estimated instead of a single model for each specific system.
- The meta model is trained offline once, using a potentially infinite stream of simulated trajectories generated by different dynamical systems.
- At inference time, the pre-trained meta model is fed with a single input/output sequence (referred to as context) from a real system, together with a query input sequence. The meta-model provides predictions of the query output sequence in a zero-shot fashion, thereby eliminating the need to implement any system identification algorithm and avoiding the selection of

$^\star$ The material in this paper was partially presented at the 20th IFAC Symposium on System Identification (SYSID), July 17–19, 2024, Boston, MA, USA, and at the 63rd IEEE Conference on Decision and Control (CDC), December 16–19, 2024, Milano, Italy.

$^{\star\star}$This work was partly supported by the Hasler Foundation project "FALCON - Foundation models for zero-shot adaptive control and prediction for dynamical systems".

*Email addresses:* matteo.rufolo@supsi.ch (Matteo Rufolo), dario.piga@supsi.ch (Dario Piga), filippo.pura@supsi.ch (Filippo Pura), marco.forgione@supsi.ch (Marco Forgione).

model structure and hyper-parameters.

The proposed meta learning approach is analogous to *Large Language Models* (LLMs) like chatGPT, LLaMA, Gemini, and DeepSeek, which are pre-trained on vast text corpora to understand and generate text. Similarly, our meta-learning approach enables users to generate output predictions by providing context and a related query, leveraging knowledge acquired in pre-training to efficiently handle various system identification tasks.

### *1.2 Related Literature*

Meta learning, also referred to as learning to learn, is a branch of machine learning introduced in the late 1980s [26] that has gained significant momentum in recent years. In a broad sense, the goal of meta learning is to enhance model and algorithm efficiency by leveraging experience from multiple related tasks. While standard machine learning trains models for a single task, meta learning approaches aim to capture common patterns and similarities across tasks, enabling rapid adaptation to new ones with minimal data and computational resources.

One of the most popular meta learning algorithms is *Model-Agnostic Meta Learning* (MAML) [10], which aims to find model parameters that can quickly adapt to new tasks. MAML has proven particularly effective in few-shot learning scenarios across various domains and applications, including computer vision for few-shot image classification tasks [2], reinforcement learning to accelerate policy learning in new environments [10], materials science for predicting material properties and accelerating materials discovery [34], as well as in system identification, optimization and control [15, 5, 24, 4].

Another instance of meta learning is represented by in-context learning, where large models are trained on diverse and extensive datasets, enabling them to process a context–a demonstration of the task at hand–along with a query input, and directly predict the corresponding output without further training [16, 7]. In this framework, zero-shot prediction emerges as a powerful feature of in-context learning, where models leverage their extensive pre-training on diverse datasets to adapt to new tasks without requiring additional fine-tuning or retraining. Typical application of in-context learning include Natural Language Processing [33], vision [31], speech recognition [6], and robotics [32], just to cite a few. In the field of system identification, in-context learning has recently demonstrated promising results for zero-shot prediction, leveraging knowledge gained from estimating the behavior of systems similar to the target one [12, 9].

### *1.3 Paper Contribution*

This paper builds upon the preliminary work in [12] on in-context learning for system identification and presents the following recent advances in the field:

- We formulate a probabilistic estimation problem, where the meta model provides not a single point estimate of the system's output, but rather the parameters characterizing a probability distribution of the output.
- We present two techniques to handle long context sequences, addressing a major limitation of [12], where the context length was limited to roughly a few thousand samples due to the quadratic growth of computational complexity.
- We show how our approach can be used to generate synthetic data for classic system identification tasks.

Preliminary versions of some of the topics covered in this paper are discussed in the conference and ArXiv preprint [22, 25]. Besides extending these previous contributions, in this paper, we provide a unified treatment of the topic, along with extensive analysis through examples and use cases, including real-world benchmarks for system identification.

The rest of the paper is organized as follows. In Section 2, we review the meta modeling framework originally introduced in [12]. We first formulate a point estimation problem of system outputs as in [12], and then extend this formulation to the more general case of probabilistic predictions. Model architectures and training algorithms are discussed in Section 3, followed by novel approaches to manage long context sequences. The first one is based on sequence patching and inspired by the Vision Transformer [8] for image classification. The second one relies on an ensembling of the meta model predictions associated with different (short) context sequences. Section 4 details how the pre-trained meta model can generate synthetic data useful for classical system identification to estimate a parametric model of a dynamical system of interest when the training data are scarce or insufficiently informative. Examples showcasing the effectiveness of the meta learning paradigm are reported in Section 5. Concluding remarks are drawn in Section 6, along with directions for future research.

## 2 Meta learning framework

### *2.1 System Class*

Let us consider a class $\mathcal{S}$ of dynamical systems. This class could correspond, for example, to Linear Time-Invariant (LTI) systems, fading memory nonlinear systems, or a more restricted subset of dynamics describing specific real-world systems. Examples of such restricted classes include the set of all robotic manipulators with a maximum number of degrees of freedom or the set of all vehicles with specific characteristics, to name just a few.

We assume to have access to a (potentially infinite) stream of input/output datasets generated by different systems $\{S^{(i)}, i = 1, 2, \dots\}$ extracted from a probability distribution $p(S)$ defined over the class $\mathcal{S}$. For each system $S^{(i)}$, a context dataset $(u^{(i)}_{1:m}, y^{(i)}_{1:m})$ and a query dataset $(\tilde{u}^{(i)}_{1:n}, \tilde{y}^{(i)}_{1:n})$ are available, with $u_k, \tilde{u}_k \in \mathbb{R}^{n_u}$, $y_k, \tilde{y}_k \in \mathbb{R}^{n_y}$. These two datasets are obtained by processing two randomly generated input sequences $u^{(i)}_{1:m}, \tilde{u}^{(i)}_{1:n}$ through the same system $S^{(i)}$, starting from different random initial conditions.

The systems $S^{(i)}$ may be instances of a high-fidelity simulator of a process of interest, with physical parameter randomly distributed according to domain knowledge. Overall, we assume there is an underlying *distribution* $p(D)$ of datasets generated by the system distribution $p(S)$ and the random input distribution. The distribution $p(D)$ may not be explicitly known, but samples $\{D^{(i)} = (u^{(i)}_{1:m}, y^{(i)}_{1:m}), (\tilde{u}^{(i)}_{1:n}, \tilde{y}^{(i)}_{1:n}), i = 1, 2, \dots\}$ can be drawn from $p(D)$.

### 2.2 *Classic system identification* vs *meta modeling*

Classic system identification aims to estimate a different model for each system $S^{(i)}$ based on the available dataset $D^{(i)}$. A classic approach might require choosing a model structure and a learning algorithm, using the context data $(u^{(i)}_{1:m}, y^{(i)}_{1:m})$. Then, the system state at a time step $c < n$ of the query may be estimated from the input/output samples $(\tilde{u}^{(i)}_{1:c}, \tilde{y}^{(i)}_{1:c})$ up to time $c$. By simulating the identified system model starting from the estimated initial condition and for the remaining query input sequence $\tilde{u}^{(i)}_{c+1:n}$, one could make predictions $\hat{y}^{(i)}_{c+1:n}$ of the output $y^{(i)}_{c+1:n}$:

$$\hat{y}^{(i)}_{c+1:n} = \mathcal{M}(\tilde{u}^{(i)}_{c+1:n}, \tilde{u}^{(i)}_{1:c}, \tilde{y}^{(i)}_{1:c}, u^{(i)}_{1:m}, y^{(i)}_{1:m}), \tag{1}$$

where $\mathcal{M}$ represents the combined identification, system state estimation, and simulation procedure applied to the dataset $D^{(i)}$.

In the meta learning approach discussed in this paper, a single meta model is estimated for all systems in the class $\mathcal{S}$. Given a context dataset generated by a specific system $S^{(i)}$ and a query input, the meta model directly provides an estimate of the corresponding query output. To achieve this goal, we introduce a (black-box) map $\mathcal{M}_\phi$, with learnable parameters $\phi$, that processes the same data as in (1), i.e.,

$$\hat{y}^{(i)}_{c+1:n} = \mathcal{M}_\phi(\tilde{u}^{(i)}_{c+1:n}, \tilde{u}^{(i)}_{1:c}, \tilde{y}^{(i)}_{1:c}, u^{(i)}_{1:m}, y^{(i)}_{1:m}). \tag{2}$$

The map $\mathcal{M}_\phi$ is trained to behave as a meta model for the system class $\mathcal{S}$ by minimizing the expected simulation error over the dataset distribution $p(D)$ in an end-to-end supervised manner, i.e.,

$$\phi^* = \arg\min_\phi \overbrace{\mathbb{E}_{p(D)}\left[\|\tilde{y}_{c+1:n} - \mathcal{M}_\phi(X)\|^2\right]}^{=J(\phi)}, \tag{3}$$

where $J(\phi)$ is the loss function, and $X = (\tilde{u}_{c+1:n}, \tilde{u}_{1:c}, \tilde{y}_{1:c}, u_{1:m}, y_{1:m})$ is introduced to compact the notation and represents all the input variables to the meta model $\mathcal{M}_\phi$.

The expectation over $p(D)$ is approximated by a Monte Carlo average over $b$ system instances from the class $\mathcal{S}$ and corresponding measured context $(u^{(i)}_{1:m}, y^{(i)}_{1:m})$ and query sequences $(\tilde{u}^{(i)}_{1:n}, \tilde{y}^{(i)}_{1:n})$, i.e.,

$$J(\phi) \approx \frac{1}{b}\sum_{i=1}^{b} \|\tilde{y}^{(i)}_{c+1:n} - \mathcal{M}_\phi(X^{(i)})\|^2. \tag{4}$$

### 2.3 *Probabilistic Prediction*

The meta model (2) generates point estimates of future outputs, which, while valuable, lack the expressiveness of a full probabilistic distribution that would provide insight into the predictive uncertainty. To address this limitation, the meta model $\mathcal{M}_\phi$ is modified to provide, at each time step, the parameters of a probability distribution of the future outputs.

Training of the meta model $\mathcal{M}_\phi$ is then performed with the aim of approximating the unknown conditional probability distribution $p(\tilde{y}_{c+1:n}|X)$ with a parameterized distribution $q_\phi(\tilde{y}_{c+1:n}|X)$. To this aim, we introduce the Kullback-Leibler (KL) divergence $\mathrm{KL}(p \parallel q_\phi)$ between $p$ and $q_\phi$:

$$\mathrm{KL}(p \parallel q_\phi) = \mathbb{E}_{p(\tilde{y}_{c+1:n}|X)}\left[\log \frac{p(\tilde{y}_{c+1:n}|X)}{q_\phi(\tilde{y}_{c+1:n}|X)}\right]. \tag{5}$$

The parameters $\phi$ characterizing the meta model $\mathcal{M}_\phi$ (and thus the approximate distribution $q_\phi(\tilde{y}_{c+1:n}|X)$) are chosen to minimize the expected KL divergence over the conditioning variables $X$, namely:

$$\begin{aligned} E_{p(X)}\left[\mathrm{KL}(p \parallel q_\phi)\right] &= E_{p(D)}\left[\log \frac{p(\tilde{y}_{c+1:n}|X)}{q_\phi(\tilde{y}_{c+1:n}|X)}\right] \\ &= \overbrace{E_{p(D)}[-\log q_\phi(\tilde{y}_{c+1:n}|X)]}^{J(\phi)} + K, \end{aligned} \tag{6}$$

where $J(\phi)$ is the loss function minimized with respect to $\phi$, and $K$ is a constant independent of $\phi$.

As for the case of point estimate discussed in the previous section, the expected loss $J(\phi)$ is approximated with the sample average:

$$J(\phi) \approx \frac{1}{b}\sum_{i=1}^{b} -\log q_\phi(\tilde{y}^{(i)}_{c+1:n}|X^{(i)}), \tag{7}$$

where $\tilde{y}^{(i)}_{c+1:n}$ and $X^{(i)}$ (with $i = 1,\dots,b$) are samples from $p(D)$ associated with the system $S^{(i)}$.

In this paper, we parameterize the distribution $q_\phi$ as a multivariate Gaussian with a diagonal covariance matrix. For notation simplicity, we present the case of single-output systems (i.e., $n_y = 1$). Thus, $q_\phi$ is factorized as:

$$q_\phi(\tilde{y}_{c+1:n}|X) = \prod_{k=c+1}^{n} q_\phi(\tilde{y}_k|X), \tag{8a}$$

$$q_\phi(\tilde{y}_k|X) = \mathcal{N}(\tilde{y}_k;\, \mu_k(X,\phi), \sigma_k(X,\phi)^2) \tag{8b}$$

where $\mathcal{N}(\cdot;\,\cdot,\cdot)$ denotes the probability density function of the univariate Gaussian distribution, with the second and third arguments representing the mean and variance, respectively. The vectors of means $\mu_{c+1:n}$ and standard deviations $\sigma_{c+1:n}$, both of dimension $n-c$, are the outputs of the meta model $\mathcal{M}_\phi$ when fed by the query input $u_{c+1:n}$, initial input/output measurements $\tilde{u}_{1:c}, \tilde{y}_{1:c}$, and context data $u_{1:m}, y_{1:m}$:

$$\mu_{c+1:n}, \sigma_{c+1:n} = \mathcal{M}_\phi(\tilde{u}_{c+1:n}, \tilde{u}_{1:c}, \tilde{y}_{1:c}, u_{1:m}, y_{1:m}). \tag{9}$$

Notably, the vector $\mu_{c+1:n}$ corresponds to the previously provided point estimate $\hat{y}_{c+1:n}$, while $\sigma_{c+1:n}$ supplies additional insight into the meta model's predictive uncertainty.

The meta model $\mathcal{M}_\phi$ is expected, intuitively, to work as a system identification algorithm, capable of extracting the dynamics of a given system $S^{(i)}$ from the provided input/output context sequence $(u^{(i)}_{1:m}, y^{(i)}_{1:m})$. Exploiting this knowledge, the meta-model generates a prediction, with the associated uncertainty estimate $\mu_{c+1:n}, \sigma_{c+1:n}$ for the query segment. Once trained, $\mathcal{M}_\phi$ will be able to generate predictions for a new, unseen system based solely on an input/output context sequence and a query input $\tilde{u}_{c+1:n}$, without requiring a model estimation for that specific system.

### *2.4 Meta model architecture and training*

The meta model is implemented using an encoder-decoder Transformer architecture similar to the one introduced in [28] for language translation, see Fig. 1 for a visual representation. With respect to [28], a recurrent patching network, described in details in Section 3.1, transforms the context data $(u_{1:m}, y_{1:m})$ into a shorter sequence $p_{1:M}$, with $p_i \in \mathbb{R}^{d_{\text{model}}}$ and $M = \frac{m}{L}$, where $L$ is an integer greater than 1. The encoder backbone then processes $p_{1:M}$ and maps it into $\zeta_{1:M}$, with $\zeta_i \in \mathbb{R}^{d_{\text{model}}}$(left side of Fig. 1). Then, the decoder processes the query input $\tilde{u}_{c+1:n}$ causally in time through masked self-attention, to generate the estimated distribution parameter $\mu_{c+1:n}, \sigma_{c+1:n}$, incorporating the knowledge of $\zeta_{1:m}$ through a cross-attention mechanism. Conceptually, the encoder can be interpreted as a system identification module, where $\zeta_{1:m}$ is an implicit representation of the system, and the decoder is a simulator.

Before the decoder block (right side of Fig. 1), the initial conditions (namely, initial input/input measurements) are processed by a linear layer that maps each time step from $\mathbb{R}^{n_u+n_y}$ to $\mathbb{R}^{d_{\text{model}}}$. Similarly, the query input $\tilde{u}_{1:c}$ undergoes a separate transformation from $\mathbb{R}^{n_u}$ to $\mathbb{R}^{d_{\text{model}}}$ (bottom-right blocks in Fig. 1). These processed sequences are concatenated to form a unique sequence of length $n$, with each element having dimension $d_{\text{model}}$. Then, a positional encoding is performed to preserve the temporal structure before the sequence enter the decoder backbone of the Transformer. The final layer of the decoder (upper-right block in Fig. 1) outputs both the mean $\mu_{c+1:n}$ and standard deviation $\sigma_{c+1:n}$ of the probability distribution of the system output at each future time step $c+1,\dots,n$.

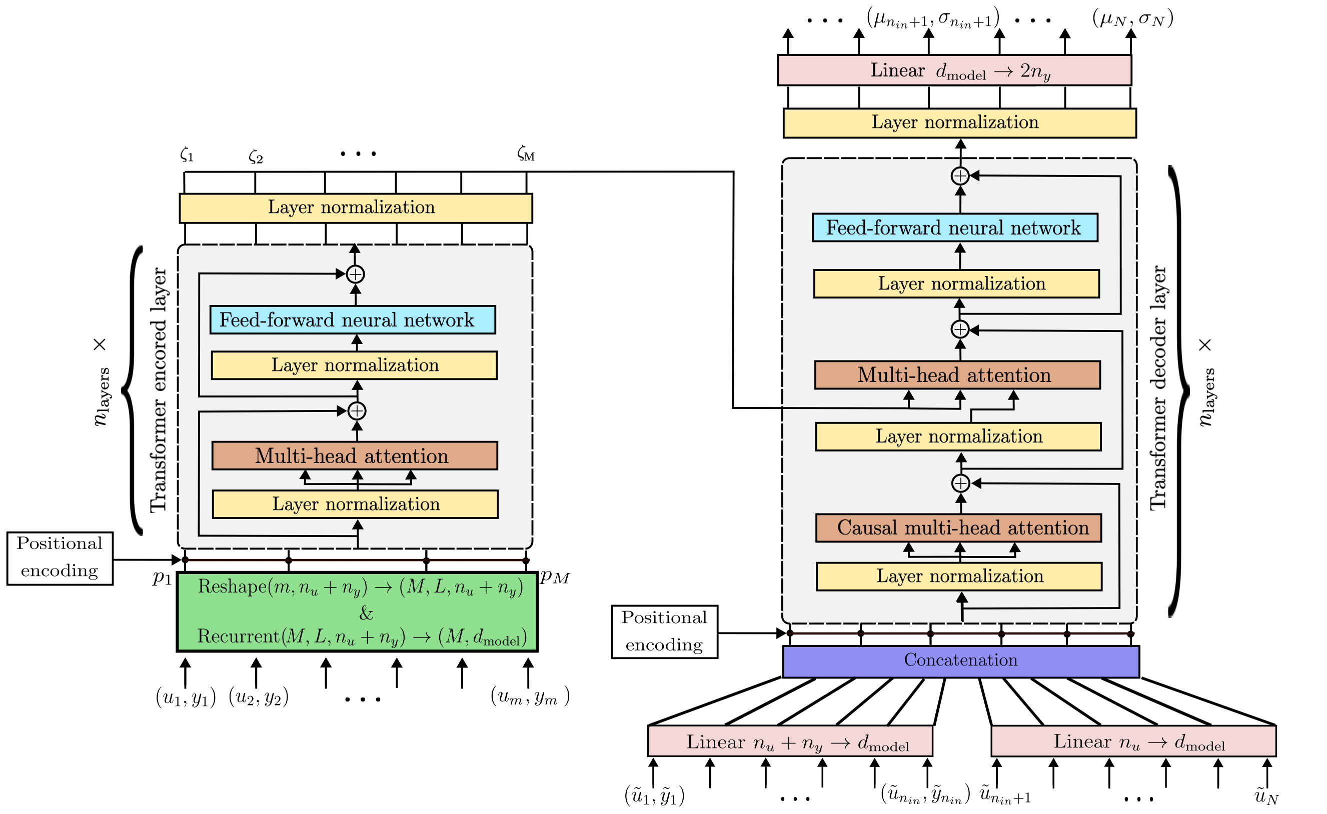
Training of $\mathcal{M}_\phi$ is performed via stochastic (or mini-batch) gradient-descent, minimizing the loss $J(\phi)$ in (7). At each iteration, a batch of $b$ systems $S^{(i)}$ (with $i = 1,\dots,b$) is drawn from the class $p(S)$ and the corresponding dataset $D^{(i)}$ is generated to compute the loss.

## 3 Handling long sequences

An intrinsic limitation of the Transformer architecture is that the attention mechanism has a computational complexity that grows quadratically with the sequence length. This results in a limitation on the length of context sequences that the meta model $\mathcal{M}_\phi$ can efficiently handle. In this section, we propose two different approaches to address these limitations: one is applied during training, and the other during inference.

### *3.1 Patching*

The patching approach, originally introduced in [8] for image processing, was later adapted for time series data in [19]. This method partitions the context sequence into smaller sub-sequences, referred to as *patches*, which are individually processed by a *patching network* to reduce the time dimensionality. This approach significantly reduces the computational cost of training by shortening the sequence length before processing by the



$\dots$ $(\mu_{n_{in}+1}, \sigma_{n_{in}+1})$ $\dots$ $(\mu_N, \sigma_N)$
Linear $d_{\text{model}} \to 2n_y$
Layer normalization
Feed-forward neural network
Layer normalization
Multi-head attention
Layer normalization
Causal multi-head attention
Layer normalization
Transformer decoder layer
$n_{\text{layers}} \times$
Positional encoding
Concatenation
Linear $n_u + n_y \to d_{\text{model}}$
Linear $n_u \to d_{\text{model}}$
$(\tilde{u}_1, \tilde{y}_1)$ $\dots$ $(\tilde{u}_{n_{in}}, \tilde{y}_{n_{in}})$ $\tilde{u}_{n_{in}+1}$ $\dots$ $\tilde{u}_N$
$\zeta_1$ $\zeta_2$ $\dots$ $\zeta_M$
Layer normalization
Feed-forward neural network
Layer normalization
Multi-head attention
Layer normalization
Transformer encored layer
$n_{\text{layers}} \times$
Positional encoding
$p_1$ $p_M$
Reshape$(m, n_u + n_y) \to (M, L, n_u + n_y)$ & Recurrent$(M, L, n_u + n_y) \to (M, d_{\text{model}})$
$(u_1, y_1)$ $(u_2, y_2)$ $\dots$ $(u_m, y_m)$

Fig. 1. Encoder-decoder Transformer for in-context system identification. The main differences with respect to the original Transformer architecture [28] are highlighted in blue and pink. Additionally, an extra layer preceding the encoder backbone, referred to as recurrent patching neural network and highlighted in green, is incorporated to manage long context sequences.

Transformer. Specifically, the input/output context sequence $(u_{1:m}, y_{1:m})$ is divided into $M$ non-overlapping patches,[1] each of length $L = \frac{m}{M}$ and dimension $n_u + n_y$. These patches are then processed by a recurrent neural network (RNN), which maps each patch to a single vector of dimension $d_{\text{model}}$. The resulting *patch embedding* sequence, $p_{1:M}$, is enriched with positional encoding to take into account temporal order information before being passed to the encoder backbone (bottom-left green block in Fig. 2).

This approach reduces the sequence length processed by the multi-head attention mechanism by a factor of $L$, leading to a computational complexity reduction of a factor $L^2$ for the training of the meta-model $\mathcal{M}_\phi$.

## *3.2 Ensembling*

Differently from patching, the ensembling method discussed in this section is applied in the inference phase, thus after meta model training, and it is based on averaging multiple predictions. It is well established that predictive performance can be enhanced through ensembling, which combines multiple models rather than relying on a single one in isolation [3, Chapter 14].

[1] For simplicity, we assume $m$ is a multiple of $M$.

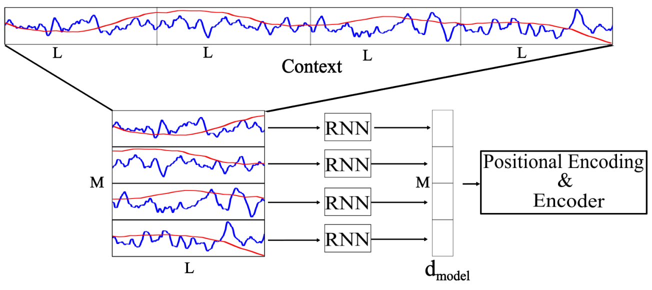


Fig. 2. Visual representation of the implemented patching approach. For clarity, a single-input ($n_u = 1$, blue) single-output ($n_y = 1$, red) system is depicted

We consider here the situation where the Transformer has been trained to process $m$-length context sequences, but at inference time a longer sequence $(u_{1:m'}, y_{1:m'})$, with $m' > m$ is available from the real system. Then, shorter $m$-length subsequences, denoted by $\{(u^i_{1:m}, y^i_{1:m}), i = 1 \dots s\}$, can be extracted from $(u_{1:m'}, y_{1:m'})$. Considering all overlapping subsequences, $s = m' - m + 1$. Each subsequence $(u^i_{1:m}, y^i_{1:m})$ is provided as context to the pre-trained Transformer, along with a fixed query input, to generate multiple predictions for the same query output, i.e.,

$$\mu^i_{c+1:n}, \sigma^i_{c+1:n} = \mathcal{M}_\phi(\tilde{u}_{c+1:n}, \tilde{u}_{1:c}, \tilde{y}_{1:c}, u^i_{1:m}, y^i_{1:m}),\ i = 1, \dots, s, \quad (10)$$

where $\mu^i_{c+1:n}$ and $\sigma^i_{c+1:n}$ represent the predicted mean and uncertainty for the query output based on the $i$-th context subsequence $(u^i_{1:m}, y^i_{1:m})$.

Finally, a single prediction is obtained by ensembling the outputs from all subsequences, improving the reliability and robustness of the estimates. In particular, the estimate at a generic time step $k$ of the query output sequence is obtained as the weighted average:

$$\mu_k = \sum_{i=1}^{s} \omega^i_k \mu^i_k. \quad (11)$$

The weights $w^i_k$ are chosen to obtain an unbiased estimate of $\tilde{y}_k$ with minimum variance, as described in the following.

From (8b), we may interpret $\mu^i_k$ as a "noisy" estimate of the true unknown output $\tilde{y}_k$:

$$p(\mu^i_k) = \mathcal{N}(\mu^i_k;\, \tilde{y}_k, (\sigma^i_k)^2). \quad (12)$$

Therefore, $\mu_k$ in (11) is also Gaussian with mean $\bar{\mu}_k = \tilde{y}_k \sum_{i=1}^{s} \omega^i_k$. The weights $\omega^i_k$ need to satisfy the constraint:

$$\sum_{i=1}^{s} \omega^i_k = 1 \quad (13)$$

to ensure that $\mu_k$ is an unbiased estimator of $\tilde{y}_k$. Furthermore, assuming that $\mu^i_k$ are independent of each others for all $i = 1, \dots, s$, the variance $\sigma^2_{\mu_k}$ of $\mu_k$ can be written:

$$\sigma^2_{\mu_k} = \sum_{i=1}^{s} (\omega^i_k \sigma^i_k)^2. \quad (14)$$

Under constraint (13), it can be proven that the variance $\sigma^2_{\mu_k}$ in (14) is minimized by the weights:

$$\omega^i_k = \frac{(\sigma^i_k)^{-2}}{\sum_{j=1}^{s} (\sigma^j_k)^{-2}}. \quad (15)$$

A detailed derivation of (15) is reported in the appendix.

Note that, in the case of $\sigma^i_k$ equal to each other for all $i = 1, \dots, s$, then $\omega^i_k = \frac{1}{s}$ and $\mu_k$ in (11) is given by an unweighted ensemble average.

## 4 Synthetic data generation for system identification

The estimated meta model $\mathcal{M}_\phi$ is able to predict, in a zero-shot learning fashion, the output of a given system of interest based on its context input/output sequence and for any possible query input. However, this predictive meta model may difficult to use for tasks beyond mere prediction due, among others, to the following reasons: the computational burden of the meta model makes it impractical for embedded applications; a parsimonious and interpretable model of the system of interest is needed, for instance to analyze system properties (such as frequency response or sensitivity to physical parameters) or for control design. In this section, we discuss how the pre-trained meta model can be anyway leveraged for classical system identification, where the goal is to estimate a (parsimonious) model $\mathbb{G}$ of a dynamical system $\mathcal{S}_\text{o}$ based on observed data.

Let us consider an observed input/output data sequence $(u_{1:m}, y_{1:m})$ (referred so far as context sequence) generated by system of interest $\mathcal{S}_\text{o}$. Since this dataset represents the available information about $\mathcal{S}_\text{o}$, it can be used either as a context for the meta model or as a training dataset for estimating a model $\mathbb{G}$ of the system $\mathcal{S}_\text{o}$, or both. For this reason, $(u_{1:m}, y_{1:m})$ will be interchangeably referred to as the "context dataset" or "training dataset", and will be denoted as $D_\text{tr}$.

We consider a parametric model $\mathbb{G}$ characterized by a parameter vector $\theta$, where $\hat{y}_t(\theta)$ represents the output of $\mathbb{G}$ for a given input sequence up to time $t$, i.e.,

$$\hat{y}_t(\theta) = \mathbb{G}(u_{1:t}; \theta). \quad (16)$$

For simplicity of exposition, we assume a single-output system.

According to classical parametric system identification, the model parameters $\theta$ are typically obtained by minimizing an average loss function $\ell$ over the training dataset $D_\text{tr}$, i.e.,

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{m} \sum_{t=1}^{m} \ell(y_t, \hat{y}_t(\theta)). \quad (17)$$

To enrich the training set $D_\text{tr}$, we aim to generate an additional, potentially infinite, set of *synthetic* input/output sequences $\tilde{u}^i_{1:\tilde{T}}, \tilde{y}^i_{1:\tilde{T}}$ where $i = 1, 2, \dots$ is used to represent the $i$-th synthetic trajectory and $\tilde{T}$ is the length of the synthetic trajectories, that for simplicity of notation, is assumed to be the same for all synthetic datasets. The synthetic trajectories are drawn from a common probability distribution $p(\tilde{u}_{1:\tilde{T}}, \tilde{y}_{1:\tilde{T}})$, ideally matching the one underlying $D_\text{tr}$.

The main idea behind the data augmentation strategy proposed in the paper is to use the pre-trained meta model $\mathcal{M}_\phi$ to generate the synthetic input/output trajectories. Indeed, the meta model has the ability to infer the behavior of the specific query system $\mathcal{S}_\text{o}$ directly from the training (or, equivalently, context) dataset $D_\text{tr}$, which implicitly encodes key characteristics of the system $\mathcal{S}_\text{o}$. This trained meta model can then predict the output $\tilde{y}_{1:\tilde{T}}$ for any synthetic input sequence $\tilde{u}_{1:\tilde{T}}$, thereby facilitating generation of a potentially infinite dimensional set of synthetic data, by implicitly transferring knowledge across systems within the same class. The underlying assumption behind the data augmentation strategies is that the target system $\mathcal{S}_\text{o}$ can be reasonably well described by the pre-trained meta model $\mathcal{M}_\phi$.

The following sections discuss two approaches for combining the training dataset $D_\text{tr}$ with synthetic data generated by the pre-trained meta model $\mathcal{M}_\phi$. These approaches are related, as both incorporate the synthetic dataset as an additional regularization term in the original loss function in (17). However, they differ in how this regularization term is constructed. In the first approach, discussed in Section 4.1 and originally introduced in [22], equal weights in the regularization term are assigned to all synthetic samples. In contrast, the second approach, presented in Section 4.2, derives the regularization term based on a maximum-likelihood formulation and considers the uncertainty of the synthetic samples in constructing the training loss.

## *4.1 Deterministic regularization*

The parameters $\theta$ that define the model $\mathbb{G}(\cdot, \theta)$ of the system $\mathcal{S}_\text{o}$ are estimated by minimizing an expected loss that incorporates both the real data $D_\text{tr}$ generated by the system $\mathcal{S}_\text{o}$ and synthetic data generated by the pre-trained meta model $\mathcal{M}_\phi$, i.e.,

$$\hat{\theta} = \arg\min_\theta \frac{1}{m}\sum_{t=1}^{m} \ell(y_t, \hat{y}_t(\theta)) + \\ + \lambda\, \mathbb{E}_{p(\tilde{u}_{1:\tilde{T}}, \tilde{y}_{1:\tilde{T}})}\left[\frac{1}{\tilde{T}}\sum_{k=1}^{\tilde{T}} \ell(\tilde{y}_k, \hat{\tilde{y}}_k(\theta))\right], \tag{18}$$

where $\hat{\tilde{y}}_k(\theta)$ is the model's estimation output for a synthetic input sequence $\tilde{u}_{1:k}$, i.e., $\hat{\tilde{y}}_k(\theta) = \mathbb{G}(\tilde{u}_{1:k}, \theta)$.

The regularization hyper-parameter $\lambda \geq 0$ controls the relative importance of real versus synthetic data. Setting $\lambda = 0$ results in training on real data only, which may cause overfitting, particularly with complex models or limited datasets. A higher $\lambda$ value emphasizes synthetic data, which can introduce distributional biases from the synthetic data generation process. Thus, $\lambda$ should be chosen based on the quantity of real training data and on the quality of the synthetic data. In this work, $\lambda$ is chosen via hold-out validation. Early stopping over a validation set is used to prevent overfitting, which may occur in case of small-size $D_\text{tr}$ and when synthetic data are not employed. To not use additional data beyond what is already used for model estimation with early stopping, the same validation set can also be used for tuning $\lambda$.

The expected value in (18) is approximated with a sample average, by applying $q$ synthetic input sequences $\tilde{u}^i_{1:\tilde{T}}$ (for $i = 1, \dots, q$) as input in the decoder, generating the corresponding outputs $\tilde{y}^i_{1:\tilde{T}}$ [2]. Consequently, the estimation problem becomes:

$$\hat{\theta} = \arg\min_\theta \frac{1}{m}\sum_{t=1}^{m} \ell(y_t, \hat{y}_t(\theta)) + \\ + \lambda \frac{1}{q}\frac{1}{\tilde{T}}\sum_{i=1}^{q}\sum_{k=1}^{\tilde{T}} \ell(\tilde{y}^i_k, \hat{\tilde{y}}^i_k(\theta)). \tag{19}$$

The optimization problem (19) can solved using minibatch stochastic gradient descent, with $q$ new synthetic input/output sequences generated at each iteration.

## *4.2 Maximum-likelihood based regularization*

In this section, we present an alternative approach based on maximum likelihood for integrating real training data $D_\text{tr}$ and synthetic data into a unified loss function for estimating the parameters $\theta$ of the model $\mathbb{G}$.

We assume that there exists an unknown "true" parameter $\theta^o$ such that, for any given input sequence $u_{1:t}$, the true system output $y^\text{o}_t$ at time $t$ is given by:

$$y^\text{o}_t = \mathbb{G}(u_{1:t}, \theta^o). \tag{20}$$

The measured training output is then modeled as:

$$y_t = \underbrace{\mathbb{G}(u_{1:t}, \theta^o)}_{y^\text{o}_t} + e_t, \quad t = 1, \dots, m, \tag{21}$$

where $e_t$ is *independent and identically distributed* (i.i.d.) Gaussian noise, i.e., $e_t \sim \mathcal{N}(0, \sigma_e^2)$.

[2] As initial conditions in the generation of the output trajectory $\tilde{y}^i_{1:\tilde{T}}$ we can consider the last $c$ samples of the context $D_\text{tr}$.

Thus, the likelihood function of the model parameters $\theta$ given the training dataset $D_{\text{tr}}$ can be factorized as:

$$p(y_{1:m}|u_{1:m},\theta) = \prod_{t=1}^{m} \mathcal{N}(y_t; \mathbb{G}(u_{1:t},\theta), \sigma_e^2). \tag{22}$$

The predicted mean provided by the meta model is used as synthetic output $\tilde{y}_k$, namely:

$$\tilde{y}_k(\tilde{u}_{1:k}, D_{\text{tr}}) = \mu_k(\tilde{u}_{1:k}, D_{\text{tr}}), \quad k = 1,\ldots,\tilde{T}, \tag{23}$$

where we emphasize the dependence of $\mu_k$ (and consequently $\tilde{y}_k$) on the training dataset $D_{\text{tr}}$ (or, equivalently, the context).

According to the meta modeling assumption in (8), the true output $\tilde{y}_k^{\text{o}} = \mathbb{G}(\tilde{u}_{1:k},\theta^o)$ of the system of interest $\mathcal{S}_o$ for a query input $\tilde{u}_{1:k}$ follows a Gaussian distribution:

$$p(\tilde{y}_k^{\text{o}}|\tilde{u}_{1:k}, D_{\text{tr}}) = \mathcal{N}(\tilde{y}_k^{\text{o}}; \mu_k, \sigma_k^2), \tag{24}$$

and

$$p(\tilde{y}_{1:\tilde{T}}^{\text{o}}|\tilde{u}_{1:\tilde{T}}, D_{\text{tr}}) = \prod_{k=1}^{\tilde{T}} \mathcal{N}(\tilde{y}_k^{\text{o}}; \mu_k, \sigma_k^2). \tag{25}$$

In the above equations, the dependence of $\mu_k$ and $\sigma_k^2$ is omitted not to burden the notation.

From the choice of the synthetic output $\tilde{y}_k$ in (22), and due to the symmetry of the Gaussian distribution, the likelihood function of the model parameters $\theta$ given the single synthetic output sequence $\tilde{y}_{1:k}$ and the training dataset $D_{\text{tr}}$ is:

$$p(\tilde{y}_{1:\tilde{T}}|\tilde{u}_{1:\tilde{T}}, D_{\text{tr}}, \theta) = \prod_{k=1}^{\tilde{T}} \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(\mu_k - \mathbb{G}(\tilde{u}_{1:k},\theta))^2}{2\sigma_k^2}\right). \tag{26}$$

Thus, the joint likelihood function of the model parameters $\theta$ given both the training and synthetic output is:

$$\begin{aligned}
&p(y_{1:m}, \tilde{y}_{1:\tilde{T}}|u_{1:m}, \tilde{u}_{1:\tilde{T}}, \theta) = \\
&p(y_{1:m}|u_{1:m},\theta) p(\tilde{y}_{1:\tilde{T}}|\tilde{u}_{1:\tilde{T}}, \underbrace{u_{1:m}, y_{1:m}}_{D_{\text{tr}}}, \theta) = \\
&\prod_{t=1}^{m} \mathcal{N}(y_t; \mathbb{G}(u_{1:t},\theta), \sigma_e^2) \prod_{k=1}^{\tilde{T}} \mathcal{N}(\tilde{y}_k; \mathbb{G}(\tilde{u}_{1:k},\theta), \sigma_k^2) = \\
&\prod_{t=1}^{m} \frac{1}{\sqrt{2\pi\sigma_e^2}} \exp\left(-\frac{(y_t - \mathbb{G}(u_{1:t},\theta))^2}{2\sigma_e^2}\right) \times \\
&\prod_{k=1}^{\tilde{T}} \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(\mu_k - \mathbb{G}(\tilde{u}_{1:k},\theta))^2}{2\sigma_k^2}\right).
\end{aligned} \tag{27}$$

Thus, the log-likelihood function is given by:

$$\begin{aligned}
&\log p(y_{1:m}, \tilde{y}_{1:\tilde{T}}|u_{1:m}, \tilde{u}_{1:\tilde{T}}, \theta) = \\
&-\sum_{t=1}^{m} \frac{(y_t - \mathbb{G}(u_{1:t},\theta))^2}{2\sigma_e^2} - \sum_{k=1}^{\tilde{T}} \frac{(\mu_k - \mathbb{G}(\tilde{u}_{1:k},\theta))^2}{2\sigma_k^2} + C,
\end{aligned} \tag{28}$$

where $C$ is a constant term that does not depend on the model parameters $\theta$.

Maximizing the log-likelihood with respect to $\theta$ is equivalent to solving the following optimization problem:

$$\min_\theta \sum_{t=1}^{m} \frac{(y_t - \mathbb{G}(u_{1:t},\theta))^2}{2\sigma_e^2} + \sum_{k=1}^{\tilde{T}} \frac{(\mu_k - \mathbb{G}(\tilde{u}_{1:k},\theta))^2}{2\sigma_k^2}. \tag{29}$$

Since the noise variance $\sigma_e^2$ is constant (and might also be unknown), problem (29) is equivalently rewritten by introducing a hyper-parameter $\lambda$ that weighs the second summation, i.e.,

$$\min_\theta \sum_{t=1}^{m} (y_t - \mathbb{G}(u_{1:t},\theta))^2 + \lambda \sum_{k=1}^{\tilde{T}} \frac{(\mu_k - \mathbb{G}(\tilde{u}_{1:k},\theta))^2}{\sigma_k^2}, \tag{30}$$

with $\lambda$ tuned through cross-validation to balance the contribution of the synthetic data term, as also discussed in Section 4.1.

The extension to the case of multiple synthetic sequences is straightforward, leading to:

$$\min_\theta \sum_{t=1}^{m} (y_t - \mathbb{G}(u_{1:t},\theta))^2 + \lambda \sum_{i=1}^{q} \sum_{k=1}^{\tilde{T}} \frac{(\mu_k^i - \mathbb{G}(\tilde{u}_{1:k}^i,\theta))^2}{(\sigma_k^i)^2}. \tag{31}$$

Summarizing, the optimization problem (31) aims at minimizing the sum of squared errors, where the first term accounts for the discrepancy between the model predictions and the observed real data, while the second term incorporates synthetic data. The synthetic data contributions are weighted inversely to their associated uncertainty, as reflected by the variance terms $(\sigma_k^i)^2$. Consequently, synthetic samples with high uncertainty have a lower influence on the loss, ensuring that the estimation process prioritizes more reliable data points.

## 5 Examples

In this section, we demonstrate the capabilities of the meta modelling paradigm through several simulation examples, as well as system identification benchmarks using real data.

Trainings and inferences are performed using an Nvidia RTX 4090 GPU with 24GB of RAM. To ensure the reproducibility of all experiments, the complete codebase, including dataset generation, training routines, and trained meta models, has been made publicly accessible in a GitHub repository [23].

## *5.1 Meta model training*

### *5.1.1 System class*

The meta model $\mathcal{M}_\phi$ is trained on synthetic datasets generated by simulating Wiener-Hammerstein (WH) systems characterized by the structure $G_1 - F - G_2$, where $F$ represents a static nonlinearity positioned between two LTI blocks, $G_1$ and $G_2$. The discrete-time LTI blocks $G_1$ and $G_2$ are randomly generated, with their orders uniformly sampled in the range $[1, 10]$, ensuring variability in their dynamic behavior. The pole locations of $G_1$ and $G_2$ are parameterized by magnitudes and phases, which are independently drawn from uniform distributions over the intervals $(0.5, 0.97)$ and $(0, \pi/2)$, respectively. The nonlinear function $F$ is implemented as a feedforward neural network with a single hidden layer that contains 32 hidden units. The network weights are drawn from a Gaussian distribution with Kaiming scaling [13].

The inputs $u_{1:m}$, used as context signals, are discrete multisine signals of length $m$, with unitary amplitude and random phase, resulting in a flat spectral density. The active frequency components are constrained within a passband defined by the spectral index range $[f_{\min}, f_{\max}]$, with $f_{\min} \geq 1$ and $f_{\max} \leq \lceil m/2 \rceil$, with $\lceil \cdot \rceil$ denoting the ceiling operator, and $\lceil m/2 \rceil$ corresponding to the Nyquist frequency for a discrete signal. The upper bound $f_{\max}$ is randomly selected from a uniform distribution over the range $[1, \lceil m/2 \rceil]$. The lower bound $f_{\min}$ is then assigned based on the following scheme:

- with probability 0.5, we set $f_{\min} = 1$, enforcing a low-pass filtering effect;
- with probability 0.5, we sample $f_{\min}$ from a uniform distribution over $[1, f_{\max}]$.

The signals are synthesized in the frequency domain and transformed into the time domain using an inverse Discrete Fourier Transform. Finally, they are normalized to maintain zero mean and unit standard deviation. Query input signals $\tilde{u}_{1:n}$ are constructed similarly.

To simulate measurement errors coming from sensors, white Gaussian noise with a standard deviation of $\sigma_{\text{noise}} = 0.01$ is added to the outputs.

### *5.1.2 Meta model architecture*

The Transformer architecture representing the meta model $\mathcal{M}_\phi$ comprises $n_{\text{layers}} = 12$ layers, each with $d_{\text{model}} = 128$ hidden units. Attention mechanisms are implemented with $n_{\text{heads}} = 4$ in both the encoder and decoder. The query input/output sequences consists $n = 130$ samples, with the initial $n_c = 30$ samples used as initial conditions.

To assess performance across different context sequence complexities, two meta models are trained: one for a context length of $m = 400$ and another for $m = 16000$. This setup enables exploration of how the length of the context sequence affects model accuracy and computational efficiency. For the extended context length of $m = 16000$, the Transformer architecture incorporates the recurrent patching approach described in Section 3.1. The context is divided into $M = 400$ patches, each of length $L = m/M = 40$. These patches are sequentially processed by a vanilla RNN with a single hidden layer with $d_{\text{model}} = 128$ units. The final hidden state of each patch is further transformed through a square linear layer of size $d_{\text{model}}$, producing a sequence $p_{1:M}$. This sequence is then used as input to the Transformer's encoder backbone. Conversely, for the shorter context length of $m = 400$, the recurrent patching method is not employed. Instead, the encoder input sequence $p_{1:M}$, where $M = m$, is directly obtained by processing the input/output samples, $u_{1:m}$ and $y_{1:m}$, through a linear layer that maps $n_u + n_y$ inputs to $d_{\text{model}}$ outputs. In both scenarios, the Transformer architecture is described by approximately 5.5 million parameters, with additional 16896 parameters for the RNN layer used in the patching strategy.

The meta model $\mathcal{M}_\phi$ is trained to minimize the negative log-likelihood loss $J_\phi$ defined in (7) across 3 million epochs, utilizing the AdamW optimizer [17] with minibatches of size $b = 32$. The training times for the meta models with context lengths of $m = 400$ and $m = 16000$ are 28 and 50 hours, respectively.

## *5.2 Meta model testing*

The predictive performance of the pre-trained meta model $\mathcal{M}_\phi$ is assessed not only across realizations of WH systems belonging to the same class used for meta model training, but also on simulated systems from a different class excited with non multisine input signals, as well as on *real* datasets from the nonlinear system identification benchmarks [1].

For the meta model handling a long context length ($m = 16000$), the patching-based approach is adopted, while ensembling is used when the meta model handling a shorter context ($m = 400$) is considered. This allows us to leverage the full context data, in case the available training dataset is larger than 400 samples.

### *5.2.1 In-distribution test data*

First, the performance of the pre-trained meta model is evaluated on test samples drawn from the same distri-

Table 1
In-distribution systems: Root Mean Square Error (RMSE) and inference time for patching-based and ensembling approach.

| Method | RMSE: mean (std) | Inference time [s] |
|---|---|---|
| Patching | 0.048 (0.044) | 0.019 |
| Ensembling | 0.038 (0.050) | 0.039 |

bution as the training data. To this aim, 320 random realizations of simulated WH systems are generated for testing.

Table 1 reports the mean and standard deviation of the Root Mean Squared Error (RMSE) achieved over the 320 realizations, along with average inference times to predict the output on a query window of length $n-c = 100$. The patching and ensembling approaches achieve similar predictive performance, with the patching approach being approximately 2 times faster in making inferences.

Figure 3 illustrates the performance of the trained meta model. For visualization purposes, only results from the patching approach are displayed. The top panel shows the true outputs, alongside the simulation errors for all WH system realizations. The bottom panel provides a detailed view of a single WH realization, showing the true output $\tilde{y}_{c+1:n}$, the predicted mean output $\mu_{c+1:n}$, the corresponding prediction error $\tilde{y}_{c+1:n} - \mu_{c+1:n}$, and the uncertainty intervals computed as $\mu_{c+1:n} \pm 3\sigma_{c+1:n}$.

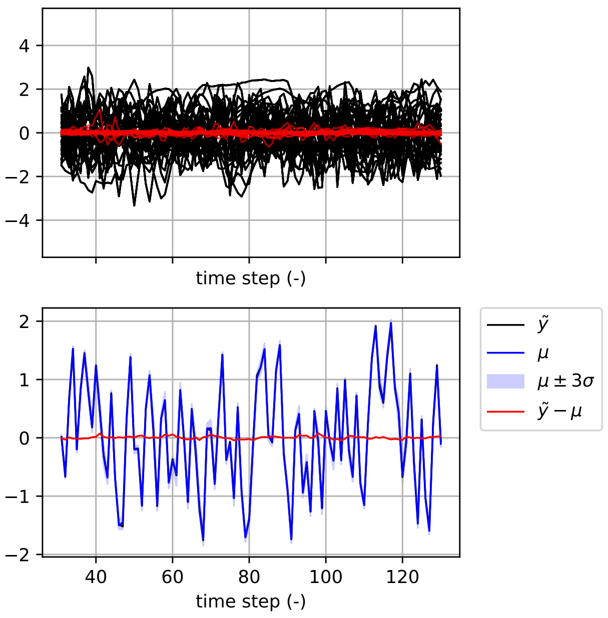


Fig. 3. In-distribution systems, patching-based approach: simulation results on 320 randomly sampled systems superposed (top) and on a single system realization (bottom). Actual output $y$ (black); simulated output mean $\mu$ (blue); simulation error $y - \mu$ (red); $\mu \pm 3\sigma$ (shaded light blue area).

### *5.2.2 Benchmarks*

The pre-trained meta model $\mathcal{M}_\phi$ is further evaluated on publicly available nonlinear system identification benchmarks, namely the WH [30], cascaded tanks with overflow [27], and coupled electric drives (CED) [29], with CED consisting of 2 input/output trajectories. The adopted benchmarks consist of *real* datasets and thus allow us to investigate the meta model's ability to generalize beyond its training domain. Notably, these datasets are used solely for testing, with no additional training performed on the benchmark-specific training data. The available training datasets serve only as context for the pre-trained meta model $\mathcal{M}_\phi$. In our analysis, we prioritize demonstrating the zero-shot capabilities of the meta modeling paradigm, rather than directly comparing predictive performance with existing system identification methodologies.

According to the benchmark guidelines, the first 50, 5, and 4 samples of the test data can be used as initial conditions for assessing predictive performance on the WH, cascaded tank, and CED benchmarks, respectively. As outlined in the previous section, the meta model is trained to utilize $c = 30$ initial conditions for inference. For the WH benchmark where 50 initial conditions are provided, only the last 30 samples are used. Conversely, for the cascade tank benchmark, zero-padding is applied to construct the sequence of initial conditions fed to the meta model. For the CED benchmark, where the training and test sequences are known to be contiguous, the sequence of initial conditions is constructed by concatenating the final 26 samples of the training data with the first 4 samples of the test.

Since the meta model is able to simulate the output over a query window of size 100, in order to accommodate the varying test sequence lengths in the benchmarks, an iterative inference approach is adopted. For the first inference step, the initial conditions provided by the benchmark dataset are used. In subsequent iterations, the $c$ final simulated output samples serve as initial conditions for the next test sub-sequence. This iterative approach, combined with the weighted ensembling strategy, significantly increases the computational time required for inference. To address this limitation, the following strategies are implemented:

- **parallelization**: the predictions for the same inference window across all 400-length context subsequences, used for the ensembling, are computed in parallel.
- **pre-computation of encoder output**: during each inference iteration, only the decoder inputs change across different iterations. Consequently, the encoder output remains constant, allowing the encoder output

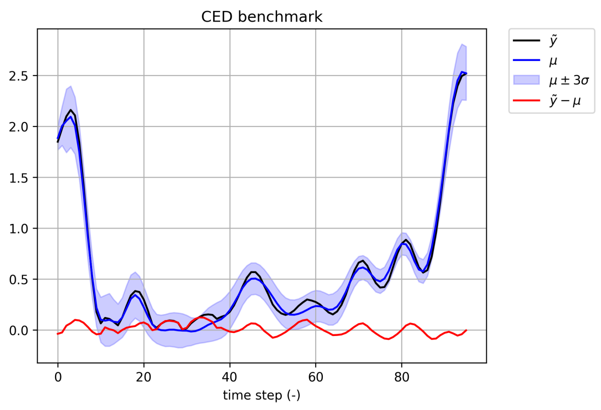


Fig. 4. CED benchmark: simulation of the second output. Actual test output $y$ (black), simulated output mean $\mu$ (blue); simulation error $y - \mu$ (red); uncertainty region $\mu \pm 3$ standard deviations (shaded blue area).

to be computed only once for all the inference iterations.

Table 2 reports the achieved RMSEs in the test datasets for the three benchmarks, along with the corresponding inference time. It is worth noting that, although the main advantage of the proposed meta modeling paradigm is its ability to perform prediction in a zero-shot learning fashion, the predictive performance is also in line with state-of-the-art system identification algorithms (see leaderboard in [20]).

Figure 4 shows the second output trajectory of the CED benchmark, where we can see the true output $\tilde{y}_{c+1:n}$ of the test dataset, the predicted mean output $\mu_{c+1:n}$ generated by the meta model; the associated prediction error $\tilde{y}_{c+1:n} - \mu_{c+1:n}$; and the 99.7% uncertainty interval computed as $\mu_{c+1:N} \pm 3\sigma_{c+1:N}$.

### 5.2.3 *Model adaptation and fine-tuning*

The performance of the pre-trained meta model is also evaluated in an out-of-distribution scenario characterized by shifts in both the input signals and the system class compared to those used for meta model training. To this end, we employed a simulator of a *continuous stirred tank reactor* (CSTR) with jacket cooling, where a first-order irreversible reaction, $A \rightarrow B$, occurs. The physical dynamics of the system are detailed in [18, Chapter 3].

Table 2
Benchmarks: RMSEs and inference time over the test datasets.

| Benchmark | RMSE | Inference time [s] |
|---|---|---|
| WH | 0.0014 | 5.91 |
| Cascaded Tanks | 0.429 | 0.22 |
| CED | [0.068, 0.055] | 0.17 |

In our analysis, the input variable is the flow rate of the cooling medium within the jackets, while the output is the concentration of reactant $A$ in the reactor. The output is normalized to have 0 mean and unitary standard deviation. The reactor temperature and the temperature of the cooling medium are considered as system states.

The input to these systems is a *random binary signal* (RBS), used to control the flow of the cooling medium inside the jackets. The context sequence length is set to $m = 16000$, leveraging the recurrent patching-based approach, while the number of initial conditions is set to $c = 30$. To emulate realistic operating conditions, white Gaussian noise with a standard deviation of $\sigma_{\text{noise}} = 0.1$ is added to the normalized outputs.

The analysis is conducted both in *zero-shot* learning and in *few-shot* adaptation setting. In the zero-shot setting, the pre-trained meta model is directly applied to predict the CSTR output. In the few-shot setting, a lightweight fine-tuning of the meta model is performed, following the methodology in [21].

For a deeper analysis, the few-shot adaptation is performed twice, under different fine-tuning settings:

- **Scenario 1**: the pre-trained meta model is adapted by only changing the input signal from multisine to RBS, maintaining the WH system class for 1000 training epochs.
- **Scenario 2**: data for meta model adaptation are collected by simulating CSTR systems with different randomly generated physical parameters, using RBS signals as input. Fine-tuning is carried out over 25 randomly generated batches of 32 systems for 1000 training epochs.

While the second fine-tuning setting is expected to yield better results as it accounts for adaptation to both the input and the system class, the first setting reflects a more realistic scenario where the user may not know the specific system class or lack access to a simulator, but has knowledge of the type of inputs that will be applied.

To test the different scenarios we generated 320 distinct realizations of the CSTR by randomly perturbing selected physical parameters of the simulator in order to introduce variability. The results of the analysis are summarized in Table 3, which includes the zero-shot case and the two few-shot cases. The lightweight fine-tuning procedure significantly enhances the generalization capabilities of the pre-trained meta model. In the first scenario, a 5x reduction in RMSE is observed, while in the second scenario the fine-tuned model achieves an RMSE of 0.115, which is close to the noise floor $\sigma_{\text{noise}} = 0.1$.

For comparison, an additional analysis is conducted, with a meta model trained from scratch directly on the

Table 3
CSTR case study: RMSE without and with fine-tuning where only the input class changes (Scenario 1) and where both input and system class change (Scenario 2) during fine-tuning .

| Setting | RMSE |
|---|---|
| Zero-shot | 1.249 |
| Scenario 1 | 0.242 |
| Scenario 2 | 0.115 |

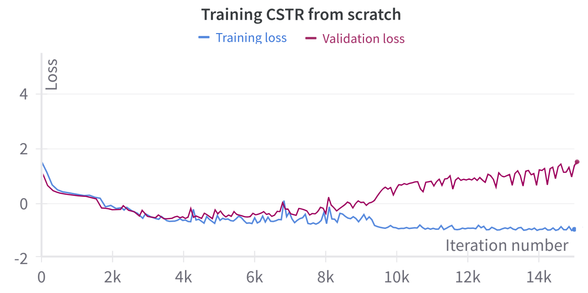


Fig. 5. CSTR case study: training the meta model from scratch on the CSTR class. Negative log-likelihood evaluated on the training and validation datasets as a function of the training iteration number.

CSTR system class. To ensure fairness, the training process utilized the same number of system realizations used for fine-tuning previously (namely, 25 batches of 32 systems). As shown in Figure 5, the training process exhibited overfitting after approximately 4000 iterations. The best-performing meta model is identified using an early stopping criterion based on the validation loss and subsequently evaluated on the same test dataset used to assess the fine-tuned models. The meta model trained from scratch achieved a test RMSE of 0.136. Notably, the performance is inferior to that obtained through fine-tuning, even though training from scratch required about 4000 iterations (against 1000 for fine-tuning) to achieve the best model in validation.

### 5.3 *Synthetic data generation for CED benchmark*

In this section, we consider the CED benchmark to demonstrate the capabilities of the pre-trained meta model in generating synthetic data, as discussed in Section 4. This benchmark is known to be challenging because of the small size of the training dataset, which consists of only 400 samples.

Since the deterministic and the maximum-likelihood-based formulations achieved similar results, we only report the ones achieved with the maximum-likelihood approach. [3]

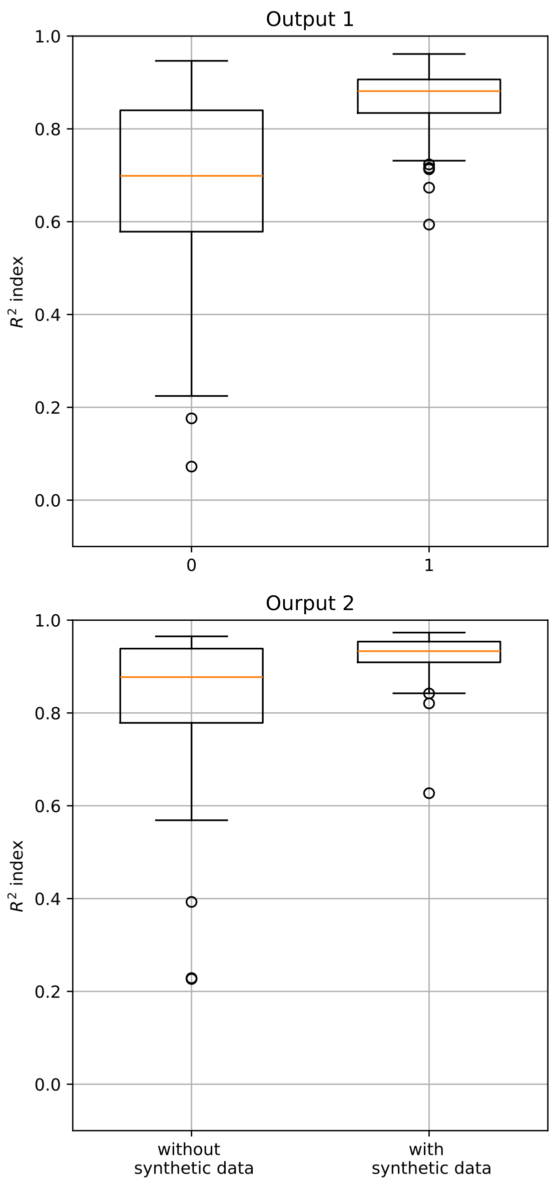


Fig. 6. CED benchmark: synthetic data generation. Boxplots of $R^2$ index on the test set with and without synthetic data.

As a parametric model $\mathbb{G}$ to describe the CED system, a dynoNet neural network [11] is employed. Its architecture comprises LTI layers, which process input sequences through filtering operations, combined with memoryless nonlinear components. Training is performed over 6000 iterations of AdamW. To mitigate overfitting, early stopping is utilized, where the first 300 samples of the training set are used to train the dynoNet, and the remaining 100 samples are used for validation. The same validation dataset is also employed to tune the regularization parameter $\lambda$ in the loss function (31). The same 300 training samples serve as a context for the pre-trained meta

[3] Results related to the determinist formulation for synthetic data generation are reported on a simulation example over the WH systems in the conference paper [22].

model $\mathcal{M}_\phi$ to generate additional 200 synthetic samples, using a multisine signal as a query input. A Monte Carlo simulation with 100 runs is conducted, where at each run, the initial weights of the dynoNet are randomly initialized, and a new synthetic query input sequence is considered.

Results on the test dataset are reported in Figure 6, which shows a boxplot of the $R^2$ index on the two outputs of the test data over the 100 Monte Carlo runs, with and without using synthetic data. These results highlight the potential of synthetic data to enhance model robustness and performance, with a median $R^2$ index of 0.70 on the first output (0.88 on the second one) without synthetic data, and a median of 0.88 on the first output (0.93 on the second one) when the synthetic dataset is used. It is also noteworthy to observe the variability of the results across the Monte Carlo runs. Indeed, when no synthetic data is used, variability is solely due to the initial condition of the dynoNet weights. On the other hand, when using synthetic data, variability arises from the initial condition of the dynoNet weights, the selected hyper-parameter $\gamma$, and the generated synthetic data. Nevertheless, less variability is observed on the achieved $R^2$ when synthetic data is used.

We remark that the synthetic data was generated by the meta-model $\mathcal{M}_\phi$, which was pre-trained on simulated WH models without any further fine-tuning or prior knowledge of the target system.

## 6 Conclusions

In this paper, we presented novel developments on in-context learning for system identification, which enables zero-shot prediction of a system of interest without the need to estimate a model for that specific system. This approach eliminates the necessity to implement traditional system identification algorithms, thereby bypassing the need to select hyper-parameters typically chosen by system identification experts, such as model type, model order, training algorithm, etc. Indeed, the meta model is trained offline using a large set of synthetic data and has the capability to directly infer the behavior of the system of interest based on the available context (namely, available training data) and then predict the system's output for any query input.

New developments based on sequence patching and ensembling also allow us to process long context sequences, circumventing the quadratic complexity of the attention mechanism of Transformers. Moreover, the meta model can generate synthetic data that can be used for data augmentation in classic system identification. This synthetic data allows to leverage information from systems in the class, as well as efficient processing of training data, implicitly learned by the encoder during the pre-training of the meta model.

Numerical results demonstrate how training a meta model with simulated data allows it to generalize well to real data or systems from a different class, potentially with quick adaptation.

Current research activities are focused on two distinct and seemingly opposite directions. On one hand, we aim at scaling to very large-scale meta models, characterized by billions of parameters. These models are expected to describe very broad classes of dynamic systems, pushing the boundaries of what dynamics meta-models can comprehend and predict. On the other hand, research is also focusing on creating distilled architectures. These lighter models aims to enhance computational efficiency and accessibility, making advanced meta models techniques more feasible for real-time applications and edge devices with limited processing capabilities.

Further developments in the field are expected to draw inspiration from advancements in Natural Language Generation algorithms, exploring the use of alternative architectures beyond Transformers, like Structured State Space and diffusion models. Additionally, there is potential to leverage open models like LLaMA or DeepSeek, applying low-rank adaptations to their weights to tailor them for the specific meta modeling challenges of dynamical systems.

## References

[1] Nonlinear benchmarks. https://www.nonlinearbenchmark.org/.

[2] Andreas Antoniou, Harry Edwards, and Amos J. Storkey. How to train your MAML. In *Proceedings of the 36th International Conference on Machine Learning*, pages 300–309, 2019.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[4] Ankush Chakrabarty, Gordon Wichern, Vedang M Deshpande, Abraham P Vinod, Karl Berntorp, and Christopher R Laughman. Meta-learning for physically-constrained neural system identification. *arXiv preprint arXiv:2501.06167*, 2025.

[5] Ankush Chakrabarty, Gordon Wichern, and Christopher R. Laughman. Meta-learning of neural state-space models using data from similar systems. In *World Congress of the International Federation of Automatic Control (IFAC)*, July 2023.

[6] Zhehuai Chen, He Huang, Andrei Andrusenko, Oleksii Hrinchuk, Krishna C Puvvada, Jason Li, Subhankar Ghosh, Jagadeesh Balam, and Boris Ginsburg. Salm: Speech-augmented language model with in-context learning for speech recognition and translation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 13521–13525, 2024.

[7] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. A survey for in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *The Eighth International Conference on Learning Representations*, 2020.

[9] Zhe Du, Haldun Balim, Samet Oymak, and Necmiye Ozay. Can transformers learn optimal filtering for unknown systems? *IEEE Control Systems Letters*, 7:3525–3530, 2023.

[10] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135, 2017.

[11] Marco Forgione and Dario Piga. dynoNet: A neural network architecture for learning dynamical systems. *International Journal of Adaptive Control and Signal Processing*, 35(4):612–626, 2021.

[12] Marco Forgione, Filippo Pura, and Dario Piga. From system models to class models: An in-context learning paradigm. *IEEE Control Systems Letters*, 7:3513–3518, 2023.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[14] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.

[15] R. Kaushik, T. Anne, and J.-B. Mouret. Fast online adaptation in robotics through meta-learning embeddings of simulated priors. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5269–5276. IEEE, 2020.

[16] Louis Kirsch, James Harrison, Jascha Sohl-Dickstein, and Luke Metz. General-purpose in-context learning by meta-learning transformers. *arXiv preprint arXiv:2212.04458*, 2022.

[17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[18] W.L. Luyben. *Chemical reactor design and control.* Wiley Interscience, 2007.

[19] Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *The Eleventh International Conference on Learning Representations*, 2023.

[20] Jean Philippe Noël, Maarten Schoukens, and Koen Tiels. Nonlinear system identification benchmarks: Result reporting. `https://www.nonlinearbenchmark.org/results-reporting`, 2025. Visited on Februray 14, 2025.

[21] Dario Piga, Filippo Pura, and Marco Forgione. On the adaptation of in-context learners for system identification. In *Proceedings of the IFAC Symposium on System Identification*, 2024.

[22] Dario Piga, Matteo Rufolo, Gabriele Maroni, Manas Mejari, and Marco Forgione. Synthetic data generation for system identification: leveraging knowledge transfer from similar systems. *arXiv preprint arXiv:2403.05164*, 2024.

[23] Filippo Pura, Matteo Rufolo, Marco Forgione, and Dario Piga. Sysid-meta-dist with patching. https://github.com/PuraFilippo/sysid_benchmarks, 2024.

[24] Spencer M. Richards, Navid Azizan, Jean-Jacques E. Slotine, and Marco Pavone. Adaptive-control-oriented meta-learning for nonlinear systems. *Proceedings of the Robotics: Science and Systems (RSS)*, 2021.

[25] Matteo Rufolo, Dario Piga, Gabriele Maroni, and Marco Forgione. Enhanced transformer architecture for in-context learning of dynamical systems. *arXiv preprint arXiv:2410.03291*, 2024.

[26] Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook.* PhD thesis, Technische Universität München, 1987.

[27] Maarten Schoukens, Per Mattsson, Torbjörn Wigren, and J Noël. Cascaded tanks benchmark combining soft and hard nonlinearities. In *Workshop on Nonlinear System Identification Benchmarks*, 04 2016.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[29] Torbjörn Wigren and Maarten Schoukens. Coupled electric drives data set and reference models. 2017.

[30] Torbjörn Wigren and Johan Schoukens. Data for benchmarking in nonlinear system identification. In *Proceedings of the 15th IFAC Symposium on System Identification*, 03 2013.

[31] Yucheng Zhou, Xiang Li, Qianning Wang, and Jianbing Shen. Visual in-context learning for large vision-language models, 2024.

[32] Jiaqiang Ye Zhu, Carla Gomez Cano, David Vazquez Bermudez, and Michal Drozdzal. InCoRo: In-Context Learning for Robotics Control with Feedback Loops. *arXiv preprint arXiv:2402.05188*, 2024.

[33] Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen, and Lei Li. Multilingual machine translation with large language models: Empirical results and analysis. *arXiv preprint arXiv:2304.04675*, 2023.

[34] Yifan Zuo et al. Accelerating materials discovery with machine learning: A review of applications and challenges. *Nature Reviews Materials*, 2021.

## Appendix

## A Minimum-variance weights for ensembling

In this appendix section we show that the variance $\sigma^2_{\mu_k}$, for a specific time step $k$, in (14) is minimized by the weights in (15), i.e.,

$$\omega_k^i = \frac{(\sigma_k^i)^{-2}}{\sum_{j=1}^s (\sigma_k^j)^{-2}}. \tag{A.1}$$

The minimal ensembling uncertainty $\sigma^2_{\mu_k}$ in (14) is given by the solution of the the optimization problem:

$$\min_{\omega_k^1,\ldots,\omega_k^s} \sum_{i=1}^s (\omega_k^i \sigma_k^i)^2. \tag{A.2}$$

subject to the constraint $\sum_{i=1}^s \omega_k^i = 1$. Using the method of Lagrange multipliers, the augmented Lagrangian is:

$$\mathcal{L}(\omega_k, \gamma) = \sum_{i=1}^s (\omega_k^i \sigma_k^i)^2 + \gamma \left( \sum_{i=1}^s \omega_k^i - 1 \right). \tag{A.3}$$

Setting the partial derivatives of $\mathcal{L}(\omega_k, \gamma)$ to zero, we obtain:

$$\frac{\partial \mathcal{L}}{\partial \omega_k^i} = 2\omega_k^i (\sigma_k^i)^2 + \gamma = 0 \implies \omega_k^i = \frac{-\gamma}{2(\sigma_k^i)^2}, \tag{A.4}$$

$$\frac{\partial \mathcal{L}}{\partial \gamma} = \sum_{i=1}^s \omega_k^i - 1 = 0 \implies \gamma = -\sum_{i=1}^s 2(\sigma_k^i)^2. \tag{A.5}$$

Substituting the expression for $\gamma$ from (A.5) into (A.4), we obtain the optimal weights in (15):

$$\omega_k^i = \frac{(\sigma_k^i)^{-2}}{\sum_{j=1}^s (\sigma_k^j)^{-2}}, \quad i = 1, \ldots, s. \tag{A.6}$$