In-context learning for model-free system identification

Marco Forgione, Filippo Pura, Dario Piga

IDSIA Dalle Molle Institute for Artificial Intelligence USI-SUPSI,
Via la Santa 1, CH-6962 Lugano-Viganello, Switzerland.

August 25, 2023

Abstract

In traditional system identification, we estimate a model of an unknown dynamical system based on given input/output sequences and available physical knowledge. Yet, is it also possible to understand the intricacies of dynamical systems not solely from their input/output patterns, but by observing the behavior of other systems within the same class? This central question drives the study presented in this paper.

In response to this query, we introduce a novel paradigm for system identification, addressing two primary tasks: one-step-ahead prediction and multi-step simulation. Unlike conventional methods, we do not directly estimate a model for the specific system. Instead, we pretrain a *meta model* that represents a class of dynamical systems. This meta model is trained from a potentially infinite stream of synthetic data, generated by systems randomly extracted from a certain distribution. At its core, the meta model serves as an implicit representation of the main characteristics of a class of dynamical systems. When provided with a brief context from a new system–specifically, a short input/output sequence–the meta model implicitly discerns its dynamics, enabling predictions of its behavior.

The proposed approach harnesses the power of Transformer architectures, renowned for their *in-context learning* capabilities in Natural Language Processing tasks. For one-step prediction, a GPT-like decoder-only architecture is utilized, whereas the simulation problem employs an encoder-decoder structure.

Initial experimental results affirmatively answer our foundational question, opening doors to fresh research avenues in system identification.

1 Introduction

In conventional system identification, researchers design algorithms that, given a dataset of input/output samples, return a model of the underlying data-generating mechanism [16, 23]. These algorithms are typically constructed to yield optimal estimates with respect to given probabilistic assumptions, or devised based on deterministic fitting arguments. The typical system identification workflow is thus closely related to supervised machine learning, with peculiarities such as the focus on dynamical systems; the choice of parsimonious and analytically tractable representations like Linear Parameter-Varying, hybrid, and piecewise-affine models [18, 4, 20]; and the use of the model for complex downstream applications such as closed-loop control [11, 22]. Given the link between supervised learning and system identification, coupled with the advancements in deep learning, recent contributions have applied modern deep learning tools to estimate dynamical systems using neural network structures [1, 9, 8, 19, 3].

In the last years, a new machine learning concept called *meta learning* has gained increased attention [14]. According to a broad definition, meta learning occurs when the problem-solving performance of a learning agent improves as more *related problems* are seen by the agent, as opposed to conventional machine learning where performance only improves as more examples from a *single problem* are seen [25]. In some sense, the goal of meta learning is thus to learn the best learning algorithm for a problem family, instead of hand-designing it with theory, intuition, or trial-and-error as done before. A typical approach is to see the algorithm design itself as a higher-level machine learning problem. In practice, this often leads to *bi-level* optimization problems where the learning algorithm is optimized (with respect to certain degrees of freedom) in the outer level and it is applied to different problem instances in the inner one [10]. For instance, the famous Model Agnostic Meta Learning (MAML) algorithm optimizes at the outer level the starting point used at the inner level for gradient-based model learning [7]. Similarly, [10] carries out the algorithm's optimization with respect to the regularization strength to be applied in the inner level. In addition to the meta learning paradigm, Large Language Models (LLMs) in Natural Language Processing embody a related concept known as *in-context learning* [6]. In this framework, a few examples serve as demonstrators and are subsequently combined with a query to form the prompt, guiding the model in generating predictions.

This paper is one of the first contributions towards the adoption of meta and in-context learning ideas and techniques for dynamical system identification. Among the few existing works, the MAML methodology has been recently applied in [5] to the estimation of Van der Pol oscillators with bounded parametric uncertainty. With respect to [5], we consider here broader classes of dynamical systems that are not necessarily close to each other in a given parameter space representation, which is a key assumption of MAML and variants thereof. To this aim, we introduce in this paper the concept of in-context learning for *model-free* system identification, where the predictions of interest are generated without deriving an explicit representation of the underlying system.

Such a model-free learning approach may be also be seen as a *meta model* that describes an entire class of dynamical systems, rather than a particular element of the class. In the developed model-free in-context learning framework, the behaviour of the actual data-generating system is "guessed" from a *context* of input/output data, and thus downstream tasks like one-step-ahead prediction or simulation are solved, which would otherwise require to learn a traditional system-specific model for each dataset. To make an analogy with LLMs, in our work, input/output trajectories generated by the underlying dynamical system play the role of in-context demonstrators. The test input trajectory acts as a query, with the predicted system's output representing the answer to that query. Like in LLMs, our guess is that training from a large amount of dynamical systems allows our "pretrained meta model" to learn latent features relevant for entire classes of dynamical systems, such as stability, hysteresis, frequency behaviour, as well as other (not easily interpretable) characteristics of these system.

Training of the meta model is performed from data generated from different but related systems. We assume to have access to an *infinite stream* of dynamical systems generating input/output datasets. In practice, data may come from simulators. We can therefore generate arbitrary amount of synthetic datasets, by varying software settings representing plausible scenarios (e.g., physical parameters and disturbances) according to physical knowledge and insights. From this stream of data, we can learn to make optimal predictions for the considered system class without modelling each individual system. The trained meta model can then be applied to make predictions on a *particular* dataset collected from a real system.

It is important to mention that model-free in-context learning of dynamical systems demands substantial representational power. This is because the meta model must address a problem whose complexity is equivalent to a system identification task. To tackle this challenge, we leverage Transformer architectures analogous to those employed in natural language processing [26, 24]. In this paper, Transformers are specialized to handle real-valued input/output sequences and address two quintessential system identification tasks: one-step-ahead prediction and multi-step-ahead simulation, using distinct Transformer architectures for the two problems. For the one-step-ahead prediction, we employ a GPT-like encoder-only structure [24], which presently stands as the state-of-the-art for natural language understanding and text generation. Conversely, for multi-step-ahead simulation, we employ a more intricate encoder-decoder Transformer, drawing inspiration from the celebrated architecture introduced in [26], which stands as the current benchmark for language translation. In our work, the encoder output can be interpreted as an implicit representation of the data-generating system, enabling the decoder to simulate the system's response to new input sequences.

As we finalized this manuscript, we came across a very recent independent work [2] that tackles the model-free one-step-ahead prediction problem using an in-context learning strategy closely mirroring ours with a GPT network. For the multi-step-ahead simulation, we are not aware of any contribution that

addresses this problem within an in-context learning framework or that uses the encoder-decoder Transformer architecture presented in this paper.

To ensure the replicability and reproducibility of our research results, and to encourage further contributions to the field, we have made the PyTorch implementation of all methodologies and results presented in this paper freely available in the GitHub repository https://github.com/forgi86/sysid-transformers. Since training of the meta-models related to the experiments reported Section 4 required about two weeks of training on a RTX 3090 GPU, we also release their trained weights in the same repository.

2 Learning framework

In traditional system identification, we are given a dataset $\mathcal{D}_{\text{train}} = (u_{1:N}, y_{1:N})$ generated by a *fixed* unknown dynamical system S, with $u_k \in \mathbb{R}^{n_u}$ (resp. $y_k \in \mathbb{R}^{n_y}$) representing the system's input (resp. output) at time step k. The objective is to estimate a model M of S from the dataset $\mathcal{D}$ and available prior assumptions on the system S, typically formalized in terms of a parametric model structure $\{M(\theta), \theta \in \Theta \subseteq \mathbb{R}^{n_\theta}\}$. The model is chosen by minimizing a cost function over the training data

$$\theta^* = \mathcal{L}(\mathcal{D}_{\text{train}}) = \arg\min_{\theta \in \Theta} d(\mathcal{D}_{\text{train}}, M(\theta)), \tag{1}$$

where d is a measure of dissimilarity between the measured data and the output's predictions of the model, such as a one-step or multi-step simulation loss. Note that in (1) we emphasized the fact that the optimal model parameter θ^* is a function $\mathcal{L}$ of the training dataset $\mathcal{D}_{\text{train}}$.

For the sake of concreteness, $M(\theta)$ could be the state-space model:

$$x_{k+1} = f_\theta(x_k, u_k) \tag{2a}$$
$$\hat{y}_k = g_\theta(x_k), \tag{2b}$$

where $x_k \in \mathbb{R}^{n_x}$ is a hidden state variable and f_θ, g_θ are functions parametrized by θ (e.g., neural networks). The cost function quantifying model fitness could be the simulation error loss:

$$d(\mathcal{D}_{\text{train}}, M(\theta)) = \frac{1}{N} \sum_{i=1}^{N} \|y_i - \hat{y}_i\|^2, \tag{3}$$

where $\hat{y}_i$ is obtained by iterating the model's equations (2) up to time step i, using the input sequence $u_{1:i-1}$. Model performance may be assessed by evaluating the dissimilarity metrics d on a distinct validation dataset $\mathcal{D}_{\text{val}}$: $d(\mathcal{D}_{\text{val}}, M(\theta^*))$, which gives an estimate of the model's *generalization error*.

In the learning framework considered in this paper and discussed in the following sections, we have a prior distribution for dynamical systems, which is used to generate a sequence of such systems. We also have a prior distribution for input signals that produces an input time sequence, exciting the generated

dynamical systems and thus resulting in a set of input-output trajectories. This allows us to generate a potentially infinite number of input-output training datasets.

Two learning frameworks are discussed in the following. First, in Section 2.1 we introduce the concept of model-based meta learning that directly produces as an output the optimized parameters of a model from given system's input/output sequence. Then, in Section 2.2 we discuss the model-free in-context learning approach, which represents the main contribution of our work.

2.1 Model-based meta learning for system identification

In the setting considered in this paper, we assume to have access to an *infinite stream* of input/output pairs $\{\mathcal{D}^{(i)} = (u_{1:N}^{(i)}, y_{1:N}^{(i)}), i = 1, 2, \dots, \infty\}$, each obtained by processing a randomly generated input signal $u_{1:N}^{(i)}$ through a randomly instantiated dynamical system $S^{(i)}$, possibly corrupted by a likewise random disturbance. In other words, we can sample (possibly synthetic) datasets $\mathcal{D}^{(i)}$ from an underlying distribution $p(\mathcal{D})$.

Since the datasets $\mathcal{D}^{(i)}$ are related with one another, partial knowledge transfer from one dataset to the other is possible, and one could then exploit it to improve the identification performance as more datasets are observed. Having access to an infinite data stream, we should be able at the limit to *learn a learning algorithm* that is optimal (w.r.t. a given criterion, e.g. one-step prediction) in some probabilistic sense (e.g., expected value) for the dataset distribution $p(\mathcal{D})$.

Let us introduce a parametrized *family of learning algorithms* $\{\mathcal{L}_\phi(\cdot), \phi \in \Phi\}$. The learning algorithm $\mathcal{L}_\phi(\cdot)$ is a map from a training dataset to a dynamical model. We seek then the learning rule that optimizes the fitting objective in an expected value sense over the dataset distribution $p(\mathcal{D})$:

$$\phi^* = \arg\min_{\phi} \mathbb{E}_{p(\mathcal{D})} \left[d\left(\mathcal{D}, \mathcal{L}_\phi(\mathcal{D})\right) \right]. \tag{4}$$

The function $\mathcal{L}_\phi$ could be either an explicit map from datasets to model parameters, e.g., a neural network from datasets to model parameters with tunable weights ϕ [13], or an *inner* optimization algorithm with tuning knobs ϕ to be applied to the dataset to obtain model parameters. A well-known example of the second approach is MAML [7], where ϕ represents the initial condition for an inner gradient-based optimization carried out on training datasets.

The *nested* nature of meta learning can be seen from the formulation in (4), where the outer objective to be minimized w.r.t ϕ is the performance index of an the inner learning algorithm $\mathcal{L}_\phi$, that in turns is designed to optimize a performance index over the training dataset $\mathcal{D}$.

Actually, the objective (4) needs to be modified to avoid the possibility of learning an overfitting algorithm, which simply memorizes each training dataset. To this end, according to the meta-learning practice [14], we can split each dataset $\mathcal{D}$ into training and validation portions $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$, respectively,

and optimize ϕ according to the criterion:

$$\phi^* = \arg\min_{\phi} \mathbb{E}_{p(\mathcal{D})} \left[d \left(\mathcal{D}_{\text{val}}, \mathcal{L}_\phi(\mathcal{D}_{\text{train}}) \right) \right], \tag{5}$$

where learning and performance evaluation of the algorithm $\mathcal{L}_\phi$ are executed on distinct splits of the dataset $\mathcal{D}$.

As known, since the expectation in (5) is intractable (the dataset distribution $p(\mathcal{D})$ may be very complex, and it may only available through samples anyway), the expectation is approximated through the sample average:

$$\mathbb{E}_{p(\mathcal{D})} \left[d \left(\mathcal{D}_{\text{val}}, \mathcal{L}_\phi(\mathcal{D}_{\text{train}}) \right) \right] \approx \frac{1}{b} \sum_{i=1}^{b} d \left(\mathcal{D}_{\text{val}}^{(i)}, \mathcal{L}_\phi(\mathcal{D}_{\text{train}}^{(i)}) \right), \tag{6}$$

where b is the sample size, and the datasets $\mathcal{D}^{(i)}$ are drawn from the probability distribution $p(\mathcal{D})$.

Having access to an infinite stream of datasets $\mathcal{D}^{(i)}$, we can actually follow a "pure" *stochastic gradient descent* approach and approximate the expected value in (5) with a sample average on b datasets resampled *at each iteration*. This is in contrast with standard supervised learning, where *gradient descent* relies on a finite number of training instances.

Having access to an infinite stream of datasets rules out the risk of *meta-overfitting*, namely of choosing a learning rule $\mathcal{L}_\phi$ too much tailored to a specific subset of datasets that does not generalize to the whole distribution.

Remark 1 *Overfitting and meta-overfitting are separate concepts corresponding to different risks. The former is related to learning a too complex $\mathcal{L}_\phi$ that tends to memorize specific properties of each dataset $\mathcal{D}_{\text{train}}^{(i)}$, without modeling the underlying mechanism $S^{(i)}$ and thus failing to generalize well to further data $\mathcal{D}_{\text{val}}^{(i)}$ from the same systems. The latter is related to learning a rule $\mathcal{L}_\phi$ too tailored to the particular datasets $\mathcal{D}^{(i)}$ seen in meta-training, which is less effective on other datasets from the same distribution $p(\mathcal{D})$. Plain overfitting may be dealt with a train-validation split of the meta objective as done in* (5)*, while meta overfitting is circumvented by training in a pure stochastic gradient descent setting, with different datasets $\mathcal{D}^{(i)}$ sampled at each iteration.*

2.2 Model-free in-context learning for system identification

In model-based meta learning, the algorithm $\mathcal{L}_\phi$ is a map taking the training dataset $\mathcal{D}_{\text{train}}^{(i)}$ as an input and returning a model $M^{(i)}$ describing the behaviour of the dynamical system $S^{(i)}$.

In the model-free in-context learning approach proposed in this paper, we learn instead a map $\mathcal{M}_\phi$ (called meta model) which processes portions of the dataset $\mathcal{D}^{(i)}$ and directly reproduces the outputs of interest, without generating an intermediate, explicit representation (namely, model) of the systems.

The two instances that are discussed and experimentally validated in this paper consist of:

- **model-free one-step-ahead prediction:** In this problem, for each input/output sequence of a dataset $\mathcal{D}^{(i)}$ and for each time step k, the meta-model $\mathcal{M}_\phi$ digests partial input/output pairs $(u^{(i)}_{1:k}, y^{(i)}_{1:k})$ up to time k and produces predictions $\hat{y}^{(i)}_{k+1}$ for the output at time step $k+1$:

$$\hat{y}^{(i)}_{k+1} = \mathcal{M}_\phi(u^{(i)}_{1:k}, y^{(i)}_{1:k}). \tag{7}$$

- **model-free simulation:** In this problem, the model model $\mathcal{M}_\phi$ receives the input/output $(u^{(i)}_{1:m}, y^{(i)}_{1:m})$ up to time step m and a test input sequence (*query*) $u^{(i)}_{m+1:N}$ from time $m+1$ to N and produces the corresponding output sequence $\hat{y}^{(i)}_{m+1:N}$.

$$\hat{y}^{(i)}_{m:N} = \mathcal{M}_\phi(u^{(i)}_{1:m-1}, y^{(i)}_{1:m-1}, u^{(i)}_{m:N}). \tag{8}$$

  We remark that in a model-based setting, this problem would be tackled by learning a system-specific model from the input/output pair $(u^{(i)}_{1:m}, y^{(i)}_{1:m})$, and then applying this model in simulation mode on $u^{(i)}_{m+1:N}$. Furthermore, note that $\mathcal{M}_\phi$ in (8) may be seen as a simulation model with structure: $M^{(i)}(\cdot) = \mathcal{M}_\phi(u^{(i)}_{1:m}, y^{(i)}_{1:m}, \cdot)$.

To solve the model-free one-step-ahead prediction and simulation problems mentioned above, the meta-model $\mathcal{M}_\phi$ is expected to "understand" (to a certain degree) the data generating mechanism $S^{(i)}$ from the provided *context* $(u^{(i)}_{1:k}, y^{(i)}_{1:k})$ (respectively, $(u^{(i)}_{1:m}, y^{(i)}_{1:m})$), but it does not return an explicit representation in a model form. Rather, as discussed in the Section 3, $\mathcal{M}_\phi$ is trained to output the one-step-ahead predictions (respectively, the output sequence continuation) of interest directly.

In the rest of the paper we focus on model-free in-context learning, which represents our core contribution and the novel system identification paradigm proposed in this work.

3 Model-free in-context learning: architectures and training

Unlike model-based meta learning, model-free in-context learning is conceptually closer to standard supervised learning where $\mathcal{M}_\phi$ is a model directly mapping from features to targets rather than a learning algorithm. Training can be seen as a single-level optimization, rather than a nested learning procedure as in (4). However, when faced with system variability in the dataset distribution, the problem setting still necessitates that $\mathcal{M}_\phi$ be as powerful as a model

learning algorithm. Therefore, in this paper, we employ Transformer architectures, which are currently state-of-the-art for in-context learning, especially in NLP applications. While other neural network architectures suitable for time sequence processing can also be utilized (like LSTM), preliminary experiments (not reported in this document) have shown that Transformer architectures consistently achieve better performance.

3.1 Decoder-only Transformer for one-step-ahead prediction

A decoder-only Transformer architecture derived from GPT-2 [24] is developed. The model structure is fully specified by the choice of the hyper-parameters characterizing a Transformer, namely: number of bottleneck layers n_{layers}; number of units in each bottleneck layer d_{model}; number of heads n_{heads}; and the context window length n_{ctx}. The standard Transformer is modified to process the real-valued input/output sequences generated by dynamical systems, instead of the sequence of symbols (word tokens) needed for natural language modeling. To this aim, with respect to plain GPT-2, the initial token embedding layer is replaced by a linear layer with $n_u + n_y$ inputs and d_{model} outputs, while the final layer is replaced by a linear layer with d_{model} inputs and n_y outputs. The overall architecture is visualized in Fig. 1.

By considering a quadratic loss on the output, the weights ϕ of the meta-model $\mathcal{M}_\phi$ are obtained by minimizing over ϕ the loss:

$$\mathbb{E}_{p(\mathcal{D})}\left[\sum_{k=1}^{N-1} \|y_{k+1} - \mathcal{M}_\phi(y_{1:k}, u_{1:k})\|^2\right], \tag{9}$$

where the expected value is approximated with b samples as

$$\mathbb{E}_{p(\mathcal{D})}\left[\sum_{k=1}^{N-1} \|y_{k+1} - \mathcal{M}_\phi(y_{1:k}, u_{1:k})\|^2\right] \approx \frac{1}{b}\sum_{i=1}^{b}\sum_{k=1}^{N-1} \left\|y_{k+1}^{(i)} - \mathcal{M}_\phi(y_{1:k}^{(i)}, u_{1:k}^{(i)})\right\|^2. \tag{10}$$

Note that the input of the meta-model $\mathcal{M}_\phi$ encompasses the entire input/output sequence from time 1 up to time $N-1$, as illustrated in Fig. 1. However, due to the causal multi-head attention layer, the output at time $k+1$ depends only on the past input/output samples up to time k. For this reason, albeit with some abuse of notation, only the sequence $u_{1:k}, y_{1:k}$ appears as the input of $\mathcal{M}_\phi$ in eqs. (9) and (10).

3.2 Encoder-Decoder Transformer for simulation

An encoder-decoder Transformer similar to the one originally introduced for language translation in [26] and adapted to the model-free in-context simulation task is used. The overall architecture is visualized in Fig. 2 and consists in: (i) an encoder that processes $u_{1:m}, y_{1:m}$ (without causality restriction) and

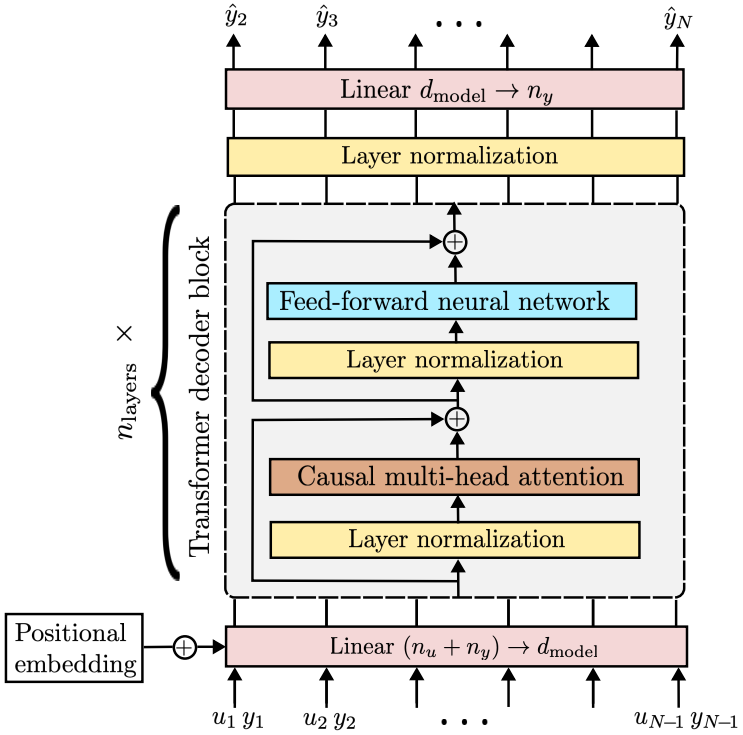


Figure 1: GPT-like decoder-only Transformer for one-step-ahead prediction. Differences w.r.t. plain GPT-2 for text generation [24, 15] are highlighted in pink.

generates an embedding sequence $\zeta_{1:m}$; (*ii*) a decoder that processes $\zeta_{1:m}$ and test input $u_{m+1:N}$ (the latter with causal restriction) to produce the sequence of predictions $\hat{y}_{m+1:N}$. Similarly to the one-step-ahead prediction task discussed in Section 3.1, the standard encoder-decoder Transformer is modified to process real-valued input/output sequences.

In a model-based interpretation, the output of the encoder $\zeta_{1:m}$ may be seen as a hidden representation of the system $S^{(i)}$ that is used as an implicit "model parameter" enabling the decoder to simulate the system's response to the sequence $u_{m+1:N}$.

Similarly to one-step-ahead prediction case, the weights ϕ of the meta-model $\mathcal{M}_\phi$ are obtained by minimizing over ϕ the loss

$$\mathbb{E}_{p(\mathcal{D})}\left[\|y_{m+1:N} - \mathcal{M}_\phi(u_{1:m}, y_{1:m}, u_{m+1:N})\|^2\right]. \tag{11}$$

As in (9), a sample-based approximation over systems $S^{(i)}$ and datasets $\mathcal{D}^{(i)}$ is

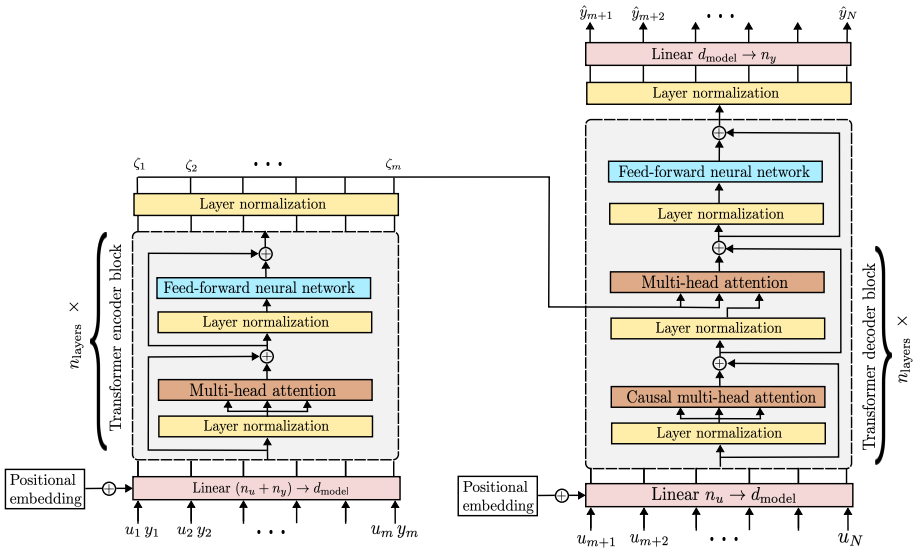


Figure 2: Encoder-decoder Transformer for multi-step-ahead simulation. Main differences w.r.t. the standard Transformer architecture for language translation [26] are highlighted in pink.

used to approximate the expected value (11):

$$\mathbb{E}_{p(\mathcal{D})}\left[\|y_{m+1:N} - \mathcal{M}_\phi(u_{1:m}, y_{1:m}, u_{m+1:N})\|^2\right] \approx \frac{1}{b}\sum_{i=1}^{b}\left\|y^{(i)}_{m+1:N} - \mathcal{M}_\phi(u^{(i)}_{1:m}, y^{(i)}_{1:m}, u^{(i)}_{m+1:N})\right\|^2. \quad (12)$$

4 Examples

In this section, we present results on model-free one-step-ahead prediction and multi-step-ahead simulation for two classes of dynamical systems, namely Linear Time Invariant (LTI) and Wiener-Hammerstein (WH). The latter represents a block-oriented description of several real-world nonlinear dynamical systems [12]. The architectures are trained by minimizing the loss functions (9) and (11) for one-step-ahead prediction and multi-step-ahead simulation, respectively. For numerical optimization, the *AdamW* algorithm [17] (a variation of plain gradient descent) is employed, and the expected value in the cost functions (9), (11) is approximated with the sample-based estimates (10) and (12), respectively, where b plays the role of the batch size.

The software has been developed in the PyTorch deep learning framework [21] and it is fully available in the GitHub repository `https://github.com/forgi86/sysid-neural-transformers`. The code of the decoder-only architecture is

adapted from the GPT-2 implementation in [15] by A. Karpathy, while we developed the code of the encoder-decoder architecture from scratch.

Computations are performed on a server of the IDSIA laboratory equipped with 2 64-core AMD EPYC 7742 Processors, 256 GB of RAM, and 4 Nvidia RTX 3090 GPUs. In all the experiments, the hardware resources of the server have been limited to 10 CPU threads and 1 GPU.

Linear Time Invariant system class (LTI). We consider random stable single-input-single-output linear time invariant dynamical systems in state-space form, with order comprised between 1 and 10. The state-space matrices A,B,C,D are randomly generated with the constraint that the eigenvalues of matrix A lie in the region within the complex unit circle with magnitude in the range $(0.5, 0.97)$ and phase in the range $(0, \pi/2)$.

Wiener-Hammerstein system class (WH) We consider stable Wiener-Hammerstein dynamical systems with structure G_1–F–G_2, where G_1, G_2 are SISO LTI blocks and F is a static non-linearity.

G_1 and G_2 are randomly generated with order comprised between 1 to 5, and with the same magnitude/phase constraint on the eigenvalues used for the LTI class introduced in the previous paragraph. The static non-linearity F is defined as a feed-forward neural network with one hidden layer and random parameters generated from a Gaussian distribution with Kaiming scaling [28].

For both system classes, the input signal applied in our experiments has a white Gaussian distribution with zero mean and unit variance. Each dataset $\mathcal{D}^{(i)}$ is thus constructed by sampling a random input sequence $u_{1:N}^{(i)}$ and by applying it to a randomly sampled system $S^{(i)}$ (either LTI or WH), thus obtaining the output sequence $y_{1:N}^{(i)}$. Finally, for easier numerical optimization, the system output $y_{1:N}^{(i)}$ is scaled to have zero mean and unit variance.

4.1 Model-free one-step-ahead prediction

For one-step-ahead prediction, we applied the decoder-only Transformer architecture described in Section 3.1, with different choices of the hyper-parameters. The results are summarized in Table 1, where we report the hyper-parameters and of the meta-model and of the training, the total number of parameters n_{param} of the meta-model, the computational time required for training, and the achieved one-step-ahead *root mean square error* (rmse).

As for the LTI system class, we train a single (small) architecture with 1.68 million parameters. Gradient-based optimization is performed over 300'000 iterations, which required about 2 hours on our server. The performance of the trained model is highlighted in Figure 3 (top row). The left plot denotes the prediction error $y - \hat{y}$ achieved by the trained Transformer over 256 randomly extracted LTI systems. It is evident that the prediction error decreases for increasing time step k, as the context $(u_{1:k-1}, y_{1:k-1})$ available to make the

Table 1: Model-free one-step-ahead prediction: settings and outcomes. The reported rmse refers to one-step-ahead prediction.

$p(\mathcal{D})$	n_{param}	n_{layers}	n_{heads}	d_{model}	n_{ctx}	n_{itr}	batch size b	train time	rmse
LTI	1.68 M	4	4	128	400	300K	32	2 h	0.01
WH	2.44 M	12	4	128	600	1M	32	0.8 d	0.04
WH	85.74 M	12	12	768	1024	10M	20	7.3 d	0.02

prediction $\hat{y}_k$ becomes larger. Around time step 20, the error's magnitude has decreased to a very small value for all the 256 visualized cases. In the right plot, we show in more detail the trajectory of a particular system, reporting the true output y, the predicted output $\hat{y}$, and the prediction error $y - \hat{y}$ altogether. It appears that, for this particular system, the Transformer is able to make nearly-optimal predictions within about 10 time steps.

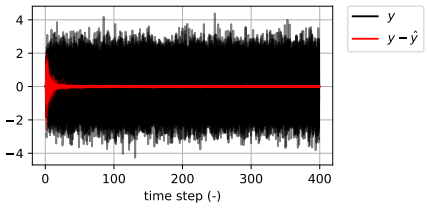
As for the WH system class, we tested a medium- and a large-size Transformer with total number of parameters 2.44 million and 85.74 million, respectively. Figure 3 (bottom row) visualizes the performance of the large-size Transformer over 32 randomly extracted WH systems. Similarly to the LTI case, we report (i) the prediction error $y - \hat{y}$ of different system realizations in the left subplot and (ii) the true output y, the predicted output $\hat{y}$, together with the prediction error $y - \hat{y}$ of one particular realization in the right subplot. We observe the same qualitative behavior previously seen in the LTI case, with the prediction error generally decreasing for increasing step index. With respect to the LTI case, the error magnitude decreases more slowly and requires a larger number of steps (about 200) to stabilize. This result is reasonable since the complexity of WH systems is much higher than the LTI class. A richer context of data is then needed to discern a particular WH system in the class, and thus to be able to make good predictions.

4.2 Model-free simulation

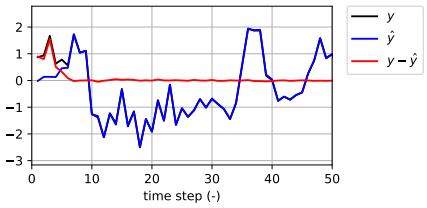
For the simulation task, we applied the encoder-decoder Transformer architecture described in Section 3.2. We generate sequences of length $N = 500$ and use the first $m = n_{\text{ctx,enc}} = 400$ samples for the encoder, and simulate over the last $N - m = n_{\text{ctx,dec}} = 100$ samples in the decoder.

The results are summarized in Table 2, where we report the hyper-parameters of the meta-model and of the training, the required computational time, and the simulation *root mean square error* (rmse). It is worth mentioning that when training the meta-model for the WH class, we had to start the optimization with the trained weights from the LTI class. Otherwise, when starting from randomly initialized weights, we observed no improvement in the training loss. This approach aligns with the rationale of *curriculum learning* [27], where tasks of increasing complexity are sequentially presented during training.

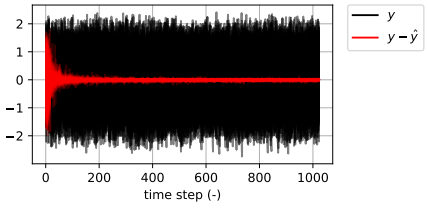
The performance of the trained meta models (both for LTI and WH classes) is visualized in Figure 4 . The left panels show the simulation error $y - \hat{y}$ achieved



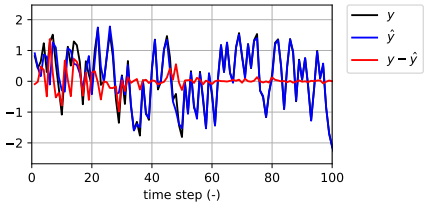
(a) Actual output (black) and prediction error (red) for 256 randomly sampled LTI systems.



(b) Actual output (black), predicted output (blue), and prediction error (red) for one randomly sampled LTI system.



(c) Actual output (black) and prediction error (red) for 32 randomly sampled WH systems.

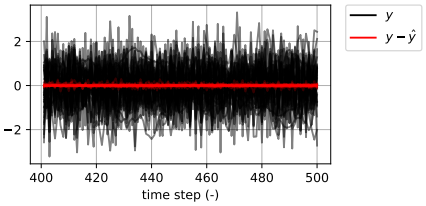


(d) Actual output (black), predicted output (blue), and prediction error (red) for one randomly sampled WH system.

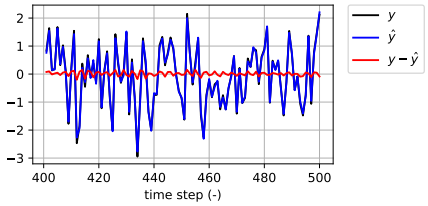
Figure 3: One-step-ahead prediction on the LTI (top row) and WH (bottom row) model classes.

Table 2: Model-free simulation: settings and outcomes. The reported rmse refers to multi-step ahead simulation. The training time for the WH class includes the time to train the LTI model class used to initialize the optimization.

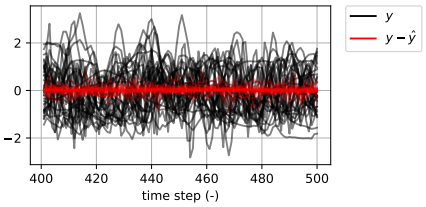
$p(\mathcal{D})$	n_{param}	n_{layers}	n_{heads}	d_{model}	$n_{\text{ctx,enc/dec}}$	n_{itr}	batch size b	train time	rmse
LTI	5.6 M	12	4	128	400/100	1M	32	0.9 d	0.04
WH	5.6 M	12	4	128	400/100	5M	32	(0.9 + 4.5) d	0.15



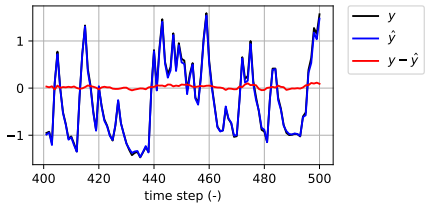
(a) Actual output (black) and prediction error (red) for 32 randomly sampled LTI systems.



(b) Actual output (black), simulated output (blue), and prediction error (red) for one randomly sampled LTI system.



(c) Actual output (black) and simulation error (red) for 32 randomly sampled WH system.



(d) Actual output (black), simulated output (blue), and simulation error (red) for one randomly sampled WH system.

Figure 4: Simulation on the LTI (top row) and WH (bottom row) model class.

by the trained Transformers over 32 randomly extracted LTI (top row) and WH (bottom row) systems. Unlike one-step-ahead prediction, the error does not decrease for increasing simulation time step. Indeed, the context $(u_{1:m}, y_{1:m})$ fed into the encoder has already being processed into the encoder embedding $\zeta_{1:m}$ before providing a query input sequence $u_{m+1:N}$. In other words, no additional information on the system behaviour can be extracted from the input-only query sequence $u_{m+1:N}$.

5 Conclusions and future works

The novel in-context learning paradigm for model-free system identification presented in this paper enables inferring the behavior of dynamical systems by observing the behavior of other systems within the same class. Compared to tradi-

tional system identification, our approach provides a training-free, few-example learning framework. This not only reduces the computational cost when adapting to new tasks, but also obviates the need for specific selections of the dynamical model structure, learning algorithm, fine-tuning of hyper-parameters, etc.

The proposed work paves the way for new research directions and raises questions that are currently being addressed by the authors and may also intrigue other researchers. The potential areas of investigation include, but are not limited to:

- **From class-to-class and from class-to-system learning:** Transfer learning techniques may be applied. For instance, the weights of a meta-model pretrained on a specific class of systems (e.g., robotic manipulators) can be adapted to describe another class (e.g., soft robots). Similarly, the meta-model pretrained on a model class can be fine tuned (or distilled) to a specific system instance.

- **Role of Noise in Learning:** It remains ambiguous whether considering noisy output data in the context during training is beneficial or counter-productive. Standard system identification suggests that noise degrades the quality of estimates. However, considering that real-world applications of this approach will involve a noisy output history, introducing noise during training might assist in learning how to filter out the noise and make accurate predictions based on a noisy context. A comprehensive statistical analysis is essential to answer this research question.

- **Curriculum Learning:** According to a curriculum learning strategy [27], the Transformer's weights can be optimized using a few-step ahead strategy. The more computationally-intensive simulation criterion can then be employed at a later stage to re-estimate only a subset of the weights. Indeed, the former is less computationally demanding, making it more suited for massive learning. Similarly, as already discussed in the example on model-free simulation (Section 4.2), we can learn a meta-model on a narrow model class (e.g., LTI systems) and then extend the training to a richer one.

- **Dynamical System Generation:** Currently, instances of dynamical systems and their corresponding input/output trajectories are drawn from an arbitrarily chosen prior distribution. There is room for the development of strategies to produce systems and signals that maximize their informativeness across successive training iterations.

- **Data Augmentation in Classic Parametric System Identification:** The proposed method can also serve as a data-augmentation technique. Here, synthetic data generated by the Transformer, based on a context generated by the system under study, can serve as additional input-output data to estimate a parsimonious parametric model of the system.

References

[1] Carl Andersson, Antônio H Ribeiro, Koen Tiels, Niklas Wahlström, and Thomas B Schön. Deep convolutional networks in system identification. In *IEEE 58th conference on decision and control (CDC)*, pages 3670–3676, 2019.

[2] Haldun Balim, Zhe Du, Samet Oymak, and Necmiye Ozay. Can transformers learn optimal filtering for unknown systems? *arXiv preprint arXiv:2308.08536*, 2023.

[3] Gerben I Beintema, Maarten Schoukens, and Roland Tóth. Deep subspace encoders for nonlinear system identification. *Automatica*, 156, 2023.

[4] Valentina Breschi, Dario Piga, and Alberto Bemporad. Piecewise affine regression via recursive multiple least squares and multicategory discrimination. *Automatica*, 73:155–162, 2016.

[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] 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.

[7] 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. PMLR, 2017.

[8] Marco Forgione and Dario Piga. Continuous-time system identification with neural networks: Model structures and fitting criteria. *European Journal of Control*, 59:69–81, 2021.

[9] 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.

[10] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1568–1577. PMLR, 10–15 Jul 2018.

[11] Michel Gevers. Identification for control: From the early achievements to the revival of experiment design. *European journal of control*, 11(4-5):335–352, 2005.

[12] Fouad Giri and Er-Wei Bai. *Block-oriented nonlinear system identification*, volume 1. Springer, 2010.

[13] Jonathan Gordon, John Bronskill, Matthias Bauer, Sebastian Nowozin, and Richard Turner. Meta-learning probabilistic inference for prediction. In *International Conference on Learning Representations*, 2019.

[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] Andrej Karpathy. nanoGPT. `https://github.com/karpathy/nanoGPT`, 2023. Accessed: June 6, 2023.

[16] Lennart Ljung. *System Identification: Theory for the User*. Prentice Hall Information and System Sciences Series. Prentice Hall, Upper Saddle River, NJ, 2nd edition, 1999.

[17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[18] Marco Lovera, Marco Bergamasco, and Francesco Casella. LPV modelling and identification: An overview. *Robust Control and Linear Parameter Varying Approaches: Application to Vehicle Dynamics*, pages 3–24, 2013.

[19] Daniele Masti and Alberto Bemporad. Learning nonlinear state–space models using autoencoders. *Automatica*, 129, 2021.

[20] Manas Mejari, Vihangkumar V Naik, Dario Piga, and Alberto Bemporad. Identification of hybrid and linear parameter-varying models via piecewise affine regression using mixed integer programming. *International Journal of Robust and Nonlinear Control*, 30(15):5802–5819, 2020.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[22] Dario Piga, Marco Forgione, Simone Formentin, and Alberto Bemporad. Performance-oriented model learning for data-driven MPC design. *IEEE control systems letters*, 3(3):577–582, 2019.

[23] Rik Pintelon and Johan Schoukens. *System identification: a frequency domain approach*. John Wiley & Sons, 2012.

[24] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[25] Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pages 3–17. Springer, 1998.

[26] 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.

[27] Xin Wang, Yudong Chen, and Wenwu Zhu. A survey on curriculum learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):4555–4576, 2021.

[28] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.