Meta-SysId: A Meta-Learning Approach for Simultaneous Identification and Prediction

Junyoung Park
KAIST
Junyoungpark@kaist.ac.kr

Federico Berto
KAIST
fberto@kaist.ac.kr

Arec Jamgochian
Stanford University
arec@stanford.edu

Mykel J. Kochenderfer
Stanford University
mykel@stanford.edu

Jinkyoo Park
KAIST
jinkyoo.park@kaist.ac.kr

Abstract

In this paper, we propose Meta-SysId, a meta-learning approach to model sets of systems that have behavior governed by common but unknown laws and that differentiate themselves by their context. Inspired by classical modeling-and-identification approaches, Meta-SysId learns to represent the common law through shared parameters and relies on online optimization to compute system-specific context. Compared to optimization-based meta-learning methods, the separation between class parameters and context variables reduces the computational burden while allowing batch computations and a simple training scheme. We test Meta-SysId on polynomial regression, time-series prediction, model-based control, and real-world traffic prediction domains, empirically finding it outperforms or is competitive with meta-learning baselines.

1   Introduction

Natural and engineered systems are often described by parameterizing a mathematical model and finding proper system parameters within that model class [1]. This *modeling and system identification* paradigm has made remarkable advances in modern science and engineering [2]–[4]. However, large datasets and recent advances in deep learning tools have made it possible to model a target system without explicit knowledge, relying instead on overly flexible model classes [5]–[8].

However, natural and engineered systems often change their behavior for various reasons, necessitating quick model adaptation with little data. One possible approach for addressing this issue is through meta-learning, which learns a meta-model $f_\theta(\cdot)$ that can quickly adapt to a new target system. Meta-learning methods can be classified into optimization-based and black-box methods depending on their adaptation mechanism [9]. Optimization-based methods explicitly optimize model parameters θ to find the task-adapted parameter θ' of adapted model $f_{\theta'}(\cdot)$. On the other hand, black-box methods use a context-dependent prediction model $f_\theta(\cdot\,;c)$ and employ an inference model to extract task-specific context c from new data.

In this study, we aim to model sets of systems whose behaviors are governed by common but unknown laws and where individual systems are differentiated by their context. We propose Meta-SysId, a meta-learning method to model the target system. Inspired by the modeling and identification framework, Meta-SysId considers target systems of the form $y = f_\theta(x;c)$, where θ denotes the function class shared by the target systems and c denotes system-specific context characteristics. Meta-SysId learns the function class f_θ during meta-training and solves the system identification problem through numerical optimization to find the proper system-specific context c. Meta-SysId can be seen as a hybridization of black-box and optimization-based meta-learning. To consider our
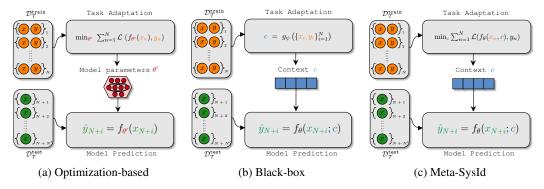
(a) Optimization-based   (b) Black-box   (c) Meta-SysId

Figure 1: Differences in meta–training and meta–testing between different meta-learning approaches. (a) Optimization–based models do not explicitly divide the parameters and context information, but optimize all model parameters online. (b) Black–box models obtain context information via the encoder network. (c) Meta-SysId obtains context information by solving an optimization problem while preserving the system class information.

target problem, it separates the roles of c and θ as done by black-box approaches. However, like optimization-based approaches, the adaptation of Meta-SysId is done through online optimization.

Moreover, the separation between θ and c provides two major advantages over other optimization-based meta-learning algorithms (e.g., [10], [11]). First, during testing, this separation reduces the burden of adaptive mechanisms, which only optimize for task-specific context c while fixing class parameters θ. In addition, this separation allows us to employ a training trick based on the exponential moving average (EMA), enabling us to train Meta-SysId with gradient descent methods, and critically without computing second-order gradients. The contributions of this work are summarized as follows:

- We propose Meta-SysId, a meta-learning approach hybridizing optimization-based and black-box methods that effectively captures shared model class information and enables online adaptability through optimization.

- We propose a simple yet effective trick for training Meta-SysId that overcomes the burden of second-order derivative calculations.

- We empirically demonstrate that Meta-SysId outperforms or is competitive to various meta-learning baselines in static function regression, time-series prediction, model-based control, and real-world traffic flow prediction domains.

2 Related works

In this section, we detail optimization-based and black-box meta-learning approaches and discuss the unique aspect of Meta-SysId to solve our targeted meta-learning problems. Fig. 1 illustrates the differences in the optimization-based and black-box approaches, and Meta-SysId.

Optimization-based meta-learning Optimization-based meta-learning methods (e.g., MAML [10], Reptile [11]) perform online optimization to find task-adapted model parameters. These approaches typically formulate the meta-learning task as bi-level optimization to find meta-parameters while considering the task-specific parameter adaptations. Different approaches solve the bi-level optimization by (1) formulating the inner optimization so that it has a closed-form solution [12], (2) differentiating the argmin via implicit differentiation [13], [14], or (3) backpropagating through the inner optimization step [10]. The first two methods limit the model class into a certain family to estimate the gradient without bias [15], [16], while the third method entails intensive memory usage during training and possibly biases the solution of the inner-level optimization [17], [18]. The separation of θ and c in Meta-SysId allows us to use the EMA trick for training, enabling us to use arbitrary network architectures (i.e., model agnosticism) while avoiding backpropagating through the inner optimization steps.

Black-box meta learning Black-box meta-learning methods (e.g., MANN [19], SNAIL [20], Neural Processes [21]–[23]) jointly meta-train an inference network $g_\psi(\cdot)$ alongside the prediction

model $f_\theta(\cdot;\cdot)$. The task-adapted predictions $y = f_\theta(x;c)$ are made using context c extracted through the inference model g_ψ. Black-box methods are easier to implement than optimization-based meta-learning methods and effectively capture the shared structure between tasks through θ. For our target problems, the role of g_ψ is to generate a solution for classical system identification. Hence, g_ψ can be viewed as an amortized optimization solver [24]. The adaptation capability of black-box methods can therefore be limited by the representation capacity of g_ψ (typically called an amortization gap). Meta-SysId also uses c to adapt, but by employing online optimization to find c, it bypasses the amortization gap.

3 Preliminaries

This section reviews classical parameter identification and meta-learning as preliminaries for explaining the proposed method.

Classical parameter identification   Consider the response y from an input x of a system f with an unknown (or unobservable) context c,

$$y = f(x;c). \tag{1}$$

Classical parameter identification aims to infer the (optimal) context c^* with data from task $\mathcal{T} = \{(x_n, y_n)\}_{n=1}^N$. Under the assumption that (x_n, y_n) pairs in $\mathcal{T}$ are generated from the same context, the identification process can be formulated as an optimization problem as follows:

$$c^* = \arg\min \sum_{n=1}^N \mathcal{L}(f(x_n;c), y_n), \tag{2}$$

where $\mathcal{L}$ is a discrepancy metric. By using the optimized context c^*, we can query responses of the (identified) system $f(\cdot;c^*)$ to inputs.

Meta-learning   Meta-learning is a collection of algorithms that learn to adapt quickly to a new task by leveraging the learning experiences from related tasks. Assuming that tasks are drawn from a distribution $p(\mathcal{T})$ and yield training data $\mathcal{D}_\mathcal{T}^{\text{tr}}$ and testing data $\mathcal{D}_\mathcal{T}^{\text{test}}$, the meta-learning objective can be defined as follows:

$$\min_{\theta,\psi} \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})} \left[\mathcal{L}\left(\mathcal{D}_\mathcal{T}^{\text{test}}, f_{\theta'}(\cdot) \right) \right] \tag{3}$$

$$\text{s.t.} \quad \theta' = g_\psi\left(\mathcal{D}_\mathcal{T}^{\text{tr}}, \theta \right), \tag{4}$$

where $g_\psi(\cdot, \theta)$ is a (meta-)learned adaptation algorithm that is parameterized by θ and ψ, $f_{\theta'}$ is the task-specific model that is parameterized by the adapted parameter θ', and $\mathcal{L}$ is a loss metric.

In summary, meta-learning uses $\mathcal{D}_\mathcal{T}^{\text{tr}}$ to learn θ and ψ such that they minimize the generalization error on $\mathcal{D}_\mathcal{T}^{\text{test}}$. Optimized meta-parameters θ^* and ψ^* can then be used to "quickly" adapt to new tasks from $p(\mathcal{T})$.

4 Meta system identification

In this section, we present meta system identification (Meta-SysId) to learn to jointly identify and predict systems in the same family. We first introduce the training problem of Meta-SysId and then discuss the technical details for efficiently solving the training problem.

4.1 Problem formulation

We formulate the learning problem of Meta-SysId as a bi-level optimization where the inner optimization Eq. (6) is for identifying the context, and the outer optimization Eq. (5) is for learning the

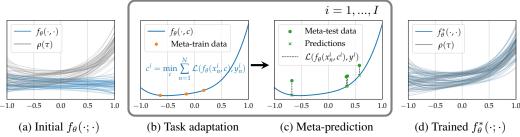(a) Initial $f_\theta(\cdot;\cdot)$ (b) Task adaptation (c) Meta-prediction (d) Trained $f_\theta^*(\cdot;\cdot)$

Figure 2: **Training of Meta-SysId** (a) the function class captured by the initial $f_\theta(\cdot;\cdot)$ and task distribution $\rho(\tau)$, (b) inference of c^i by solving Eq. (6) with meta-train data and identification of $f_\theta(\cdot, c^i)$, (c) Prediction of $f_\theta(\cdot, c^i)$ and the outer loss $\mathcal{L}(f_\theta(x_n^i; c^i), y_n^i)$, (d) the optimized $f_\theta^*(\cdot, \cdot)$. Meta-SysId is trained by repeating (b), (c), and minimizing $\mathcal{L}(f_\theta(x_n^i; c^i), y_n^i)$.

class conditioned on the identified context. The proposed optimization problem is as follows:

$$\min_\theta \quad \mathbb{E}_{\mathcal{T}^i \sim \rho(\mathcal{T})} \sum_{n=N+1}^{N+N'} \mathcal{L}\left(f_\theta(x_n^i; c^i), y_n^i\right) \tag{5}$$

$$\text{subject to} \quad c^i = \arg\min_c \sum_{n=1}^{N} \mathcal{L}\left(f_\theta(x_n^i; c), y_n^i\right), \tag{6}$$

where $\mathcal{T}^i = \{(x_n^i, y_n^i)\}_{n=1}^{N+N'}$ is the i-th task, N and N' are the meta-train and -test dataset sizes, respectively, f_θ is a prediction model parameterized by θ, and $\mathcal{L}$ is a loss metric.

By solving Eqs. (5) and (6), f_θ is trained. The trained $f_\theta^*(\cdot;\cdot)$ represents the function class that is shared by all tasks from $p(\mathcal{T})$. Identification of the specific function (i.e., task adaptation) is done by solving Eq. (6) with the data points sampled from a specific task. After finding c^i, we can query for meta-test predictions with $f_\theta(\cdot, c^i)$ as visualized in Fig. 2 (b) and (c).

Meta-SysId can be viewed as a special case of the meta-learning recipe explained in Section 3 by considering adapted parameters θ' to contain fixed class parameters θ and optimized context c^i. Even though Meta-SysId is formulated similarly to optimization-based meta-learning methods, the optimization variables of Meta-SysId are the context input c^i alone rather than the model parameters. In terms of implementation, this algorithmic selection allows us to batch-solve inner-level optimization with standard automatic differentiation tools. Meta-SysId can therefore be trained efficiently. We can interpret this clear separation of θ and c as capturing function classes and their coefficients separately. We further inspect this view of Meta-SysId in Section 5.1.

4.2 Training Meta-SysId

We use a simple-yet-effective method based on EMA to train Meta-SysId while maintaining model agnosticism and lowering memory consumption. First, we copy θ to create a delayed target model $f_{\bar{\theta}}$ parameterized by $\bar{\theta}$. We use $f_{\bar{\theta}}$ to infer c^i of Eq. (6) through gradient descent. We then optimize θ by solving Eq. (5) with c^i. As f_θ and $f_{\bar{\theta}}$ are independent, this does not require differentiating the argmin operator to calculate the loss gradient. After a gradient update, we update $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$ with $0 \leq \tau \ll 1$. The training procedure is summarized in Algorithm 1.

Intuition The training problem of Meta-SysId can be seen as a Stackelberg game where the outer optimization leads and the inner optimizations follow. In this game perspective, solving the training problem through implicit differentiation can find θ exactly. However, gradient estimation of implicit differentiation can be possibly biased when the inner problem is not solved optimally [15], [16]. On the other hand, solving the proposed training problem with the EMA trick can be interpreted as formulating the training problem as a Nash game. The training trick can be interpreted as a variant of the proximal decomposition algorithm [25], which is often employed to find Nash equilibria. We also empirically found that solving the training problem with the proposed trick is more stable during

4

Algorithm 1: Training Meta-SysId with exponential moving average (EMA)

Input: Prediction model f_θ, Tasks $\mathcal{D}_\mathcal{T}$, inner optimization steps K, inner optimization step size α, Weighting factor τ,

1   $\bar{\theta} \leftarrow \theta,\ f_{\bar{\theta}} \leftarrow f_\theta$ // Initialize the target model
2   **for** $1, 2, ...$ **do**
3   Sample batch of tasks $\mathcal{T}^i \sim \mathcal{D}_\mathcal{T}$
4   **for** *all* $\mathcal{T}^i$ **do**
5       $c^i \leftarrow 0$
6   **for** $k = 1, ..., K$ **do**
7         $\mathcal{L}(c^i) = \sum_{n=1}^N \mathcal{L}(f_{\bar{\theta}}(x_n^i; c), y_n^i)$
8         $c^i \leftarrow c^i - \alpha \nabla_{c^i} \mathcal{L}(c^i)$ // Solve Eq. (6)
9   **end**
10   **end**
11   Evaluate $l(\theta) = \sum_{\mathcal{T}^i} \sum_{n=N+1}^{N'} \mathcal{L}(f_\theta(x_n^i; c^i), y_n^i)$
12    $\theta \leftarrow \theta - \alpha \nabla_\theta l(\theta)$
13    $\bar{\theta} \leftarrow \tau \theta + (1 - \tau)\bar{\theta}$ // Update the target model
14 **end**

---

training and converges to better prediction models than implicit differentiation or backpropagation through the optimization steps. We compare the prediction results of Meta-SysId with different training methods in Section 5.1.

5   Experiments

In this section, we examine various properties of Meta-SysId with static function regression, we apply Meta-SysId for time-series prediction of a dynamical system, we show the efficacy of Meta-SysId for model-based control, and we use Meta-SysId to perform real-world traffic flow prediction.

5.1   Polynomial regression

We first examine polynomial function regression to understand the properties of Meta-SysId. In this task, the objective is to identify polynomial coefficients and predict the identified functions. We prepare 4th order polynomial datasets by sampling the coefficients from $\mathcal{U}(0.1, 2.5)$, and set N and N' as 5 and 15, respectively. The train and test datasets contain 500 and 200 polynomials, respectively. We provide the details of the model architecture, training scheme, and additional experimental results in Appendix A.1.

Do θ and c contain function class and parameter information, respectively? The role of θ and c are to capture a function class and to parameterize a specific function from that class, respectively. For example, θ should express the fact that functions are polynomials and c should capture the coefficients of a specific polynomial. To analyze whether θ and c fulfill these roles, we train two Meta-SysId models: one with parameters θ_1 meta-trained on linear functions, and one with parameters θ_2 meta-trained on quadratic functions. We then define models that interpolate between these two sets of parameters with $\theta(\lambda) := (1 - \lambda)\theta_1 + \lambda\theta_2$ with $0 \le \lambda \le 1$. In Fig. 3, we visualize predictions from $f_{\theta(\lambda)}(\,\cdot\,; c)$ with different samples of $c \sim \mathcal{U}(-0.025, 0.025)^{32}$ and different values of λ. We conjecture that $f_{\theta(\lambda)}(\,\cdot\,; c)$ represents the linear and quadratic
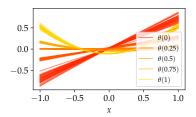

Figure 3: Role of varying contexts within model classes $\theta(\lambda)$ that interpolate between quadratic and linear.

functions, respectively, when $\lambda = 0$ and 1 and the $f_{\theta(\lambda)}(\,\cdot\,; c)$ represent the functions that interpolate the linear and quadratic functions when $0 < \lambda < 1$. We verify the conjectures by observing that within a single function class $\theta(\lambda)$, each c parameterizes a different function from that class (e.g., the red curves are all linear but each of them is a different linear function).
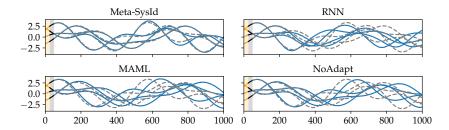
Figure 6: Mass-spring system prediction. All models infer the physical constants from historical (shaded in orange) and current (shaded in black) state observations, and then to predict future state trajectories (colored in blue). The gray dashed lines the visualize prediction targets. Meta-SysId outperforms the baselines in the 25-steps and 975-steps predictions.

Is the EMA trick an effective training scheme? Training Meta-SysId entails solving the bi-level training problem. To understand the effect of the training method on the performance of Meta-SysId, we examine the test MSE of the different training methods. Fig. 4 demonstrates the test MSE of `Meta-SysId` trained with the EMA trick (EMA), backpropagation-through-optimization (BPTO), and implicit differentiation (Implicit) over the training steps. We average the results of five independent runs. From the results, we confirm that the proposed target network method exhibits stable and better training results. On the contrary, Implicit fails to converge. We attribute this difference to a biased gradient estimate as Eq. (6) generally
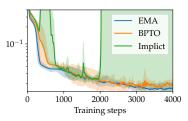

Figure 4: MSE vs. train steps for different training methods.

can not be solved optimally [15], [16]. It is also noteworthy that EMA has more stable and better performance than BPTO, which is trained end-to-end. This difference does not originate from the inner optimization itself since all models take the same number inner optimization steps, $K = 100$. We conjecture that this happens because f_θ and $f_{\bar\theta}$ are independent during θ updates. The inferred c^i acts as noise-injected context and thus training f_θ more robustly. Related work also studies the relationship between noise injection and robustness of learned models [26]–[28].

Does the optimization budget affect predictive performance? The context inference of Meta-SysId is done by solving Eq. (6). As shown in Algorithm 1, we employ gradient descent a pre-specified K times to solve Eq. (6). To analyze the effect of K to the predictive performance of Meta-SysId, we train `Meta-SysId` models with different values of K and evaluate them on the test dataset. Fig. 5 visualizes the predictive performance of `Meta-SysId` against the number of test time optimization steps. In general, `Meta-SysId` with higher K performs better at the cost of optimization time. We also observe that `Meta-SysId` can improve prediction performance by spending more optimization budget when testing. Moreover, we observe that `Meta-SysId` outperforms Atnn when the test optimization step is larger than 40, regardless of K.
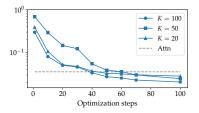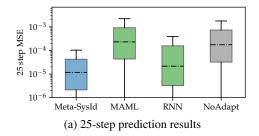

Figure 5: SE vs. test optimization steps for Meta-SysId trained with the pre-specified inner optimization steps K.

5.2 Time-series prediction

We then evaluate Meta-SysId to perform time-series prediction in mass-spring systems visualized in Fig. A.1. In this setting, the objective is first to infer the physical constants (i.e., the masses and spring constants) from historical and current state observations, and then to predict future state trajectories as shown in Fig. 6.

We consider the mass-spring system composed of two distinct masses and three springs without friction. We sample the masses and spring constants from $\mathcal{U}(0.75, 1.25)$ and generate the trajectory of the four dimensional state (i.e., the positions and velocities of masses) by numerically solving the
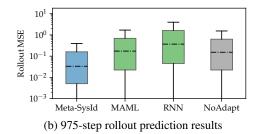
6

(a) 25-step prediction results

(b) 975-step rollout prediction results

Figure 7: Mass-spring system prediction results as box-plots showing MSE quartiles. Meta-SysId, meta-learning baselines and `NoAdapt` results are colored with blue, green, gray, respectively.

system equation for 10 seconds with 0.001 seconds intervals. Train and test datasets are composed of 100 and 50 trajectories, respectively. We set N to 25 (i.e., 0.025 seconds of history is used to infer c).

We employ `MAML` [10] as an optimization-based meta-learning baseline and `RNN` [29] as a black-box meta-learning baseline. To understand the effect of adaptation, we employ `NoAdapt`, a model that does not explicitly adapt to the target tasks. For all models, including `Meta-SysId`, we employ the same network for f_θ, which takes N recent states to predict N future states. Please refer to Appendix A.2 for architecture and training details.

We assess the predictive performance of `Meta-SysId` and the baseline models by evaluating MSE for 25-step predictions and 975-step rollout predictions on the test trajectories. As shown in Figs. 7a and 7b, `Meta-SysId` shows significantly lower N-step and rollout prediction errors than the baselines. Considering all models use the same f_θ architecture, we can conclude that the optimization-based context identification of Meta-SysId helps accurately predict states both near and far into the future.

5.3 Model-based control

To validate Meta-SysId is applicable to the model-based control, we consider the context-aware optimal control problem:

$$\min_{u_{0:T-1}} \quad \sum_{t=0}^{T-1} J\left(x_t, u_t\right)$$
$$\text{subject to} \quad x_{t+1} = f(x_t, u_t; c_t) \; \forall t \in 0, \ldots, T-1 \tag{7}$$

where the control actions u_t are optimized to minimize an appropriately chosen cost function J. In particular, we consider the case of Model Predictive Control (MPC) [30], [31] of a Planar Fully Actuated Rotorcraft (PFAR), in which the optimization is performed *online* (see Appendix A.3 for additional information on the control algorithm and system dynamics). We refer to the PFAR as a "drone" for simplicity.

We train `Meta-SysId` and `RNN` [29] for 50 epochs with the Adam optimizer [32] with learning rate of 10^{-3}. Context inference in `Meta-SysId` is performed by iterating for $K = 50$ steps, also with the Adam optimizer and a learning rate of 10^{-3}. Latent context size is set to 32 in order not to bias the model with the real context size of 1 (horizontal wind). Models are trained on trajectories with episode-specific constant wind with a time step $\Delta t = 0.02\,\text{s}$ and context inference on the past 25 time steps. See Appendix A.3.3 for details. We introduce two separate control tasks in the following paragraphs. We also considered employing a MAML-based baseline as an optimization-based method. However, training the MAML-based models is intractable to our computing infrastructure. For more details, please refer to Appendix A.3.4.

Reference trajectory tracking with contant context We perform MPC on the drone model to follow randomly generated smooth reference trajectories. In each episode, we apply the wind with the speed sampled from $\mathcal{U}(0, 8)$ m/s to the x-axis, similar to what was done in [33]. Figure Fig. 8d shows the results of MPC for a sampled trajectory and the distribution of control costs over the 1,000 sampled trajectories, with `Meta-SysId` achieving lower control cost than `RNN`. See Appendix A.3.5 for additional details.
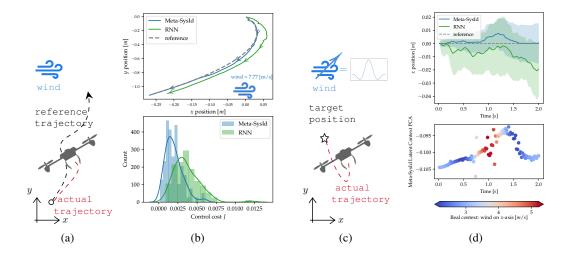
Figure 8: [Left] (a) Drone trajectory tracking task diagram. (b) Sample controlled trajectory and distribution of control costs show our `Meta-SysId` achieves better performance than the baseline. [Right] (c) Diagram for drone stabilization task with dynamic wind. (d) Unlike the baseline, `Meta-SysId` manages to keep the drone stabilized even with changing context; the inferred context visualized by 1D PCA over time at the bottom reveals smooth changes in the real context are approximately reflected in the latent space.

Table 1: PeMSD7(M) results

	MAE (15/30/45 min)	MAPE (15/30/45 min)	RMSE (15/30/45 min)
STGCN(Cheb) [36]	2.25/3.03/3.57	5.26/7.33/8.69	4.04/5.70/6.77
STGCN(1st) [36]	2.26/3.09/3.79	5.24/7.39/9.12	4.07/5.77/7.03
STGCN-Cov [37]	2.20/2.97/3.51	5.14/7.26/8.74	4.02/5.64/6.70
MAML	**2.17**/2.89/3.21	**5.06**/6.95/7.89	**3.90**/5.21/**5.81**
STGCN-Enc (black-box)	2.24/2.85/3.54	5.19/7.00/8.51	3.96/5.27/6.10
Meta-SysId	2.19/**2.79**/**3.16**	**5.06**/**6.81**/**7.84**	3.93/**5.17**/5.87

Stabilization over time–varying context   We also test the model under a time–varying context, namely a horizontal wind characterized by the profile of an *extreme operating gust* [34]. We control the drone with MPC with the task of stabilizing positions and velocities to the origin. Fig. 8d shows x-axis positions over 10 runs and a 1–dimensional PCA visualization [35] of the latent context inferred by `Meta-SysId` over time. While RNN suffers from trajectory drifting due to the varying context, `Meta-SysId` manages to compensate for the gust of wind. The latent context visualization also provides a further insight: `Meta-SysId` can infer an approximately *smooth* change in latent context which matches the real–life smooth-yet-sharp change in wind. On the other hand, as shown in Fig. A.3, RNN does not generally yield an interpretable latent context variation. Further details are available in Appendix A.3.5.

5.4   Real-world traffic flow prediction

As mentioned in Section 1, Meta-SysId is devised to solve the meta-learning tasks where each task is assumed to share a common function class and the context c differentiates tasks. However, the training of Meta-SysId can be done without these assumptions. We hypothesize that Meta-SysId learns to generate a "virtual" context and, by leveraging the learned context, can improve the performance of general time-series prediction tasks. To validate this hypothesis, we apply Meta-SysId to conduct time-series prediction on a real-world traffic dataset.

We consider the PeMSD7(M) dataset [36], which consists of traffic measurements taken at 5-minute intervals on weekdays in May and June of 2021 at 228 locations in District 7 of California. The objective is to use 60 minutes of recent measurements to predict 15, 30, and 45 minutes into the future.

We employ the spatio-temporal graph convolution network (STGCN) with the same hyperparameters [36] for $f_\theta(\cdot)$ of `Meta-SysId` with modifications to the input dimension to account for context. To compare against non-meta-learning approaches, we baseline against STGCN with Chebyshev polynomial approximations (`STGCN(Cheb)`), STGCN with first-order approximations (`STGCN(1st)`), and STGCN with Covariance Loss regularization (`STGCN-Cov`) [37]. For meta-learning baselines, we consider `MAML` [10] and `STGCN-Enc`, a model which uses STGCN as the history encoder for adaptation (i.e., black-box approach). All meta-learning approaches use the last 60-minutes of observations and their corresponding labels to infer context. Note that we can always use this strategy to infer context inputs because they are *only* composed from history. For more training details, see Appendix A.4.

Table 1 summarizes the prediction errors of the different models. We find that meta-learning approaches outperform non-meta learning baselines, but this might be the effect of using additional data (i.e., the data for adaptation). Among the meta-learning approaches, `Meta-SysId` shows competitive or leading predictive performance, which is noteworthy because it is designed to solve meta-learning problems which separate context and function class. This suggests that Meta-SysId can extract meaningful "virtual" context to enhance predictive performance when real context may not exist.

6   Conclusion

Motivated by traditional *modeling and system identification* approaches, in this work, we studied how to generalize modeling for families of tasks that have unknown but shared model structures and differ in task context. We presented Meta-SysId, an optimization-based meta-learning model that separates model class parameters from task-specific context variables. Meta-SysId treats the context as an extra input of the prediction model and infers the context variable during meta-testing through optimization. To train Meta-SysId, we employed an Expected Moving Average (EMA) trick to avoid differentiating through the inner context-finding optimization problem, resulting in a stable training procedure that relies exclusively on first-order gradients. We empirically found that Meta-SysId performs competitively or outperforms optimization-based and black-box meta-learning baselines in several domains ranging from regression to online control to real-world prediction.

A current limitation of our work is that we only consider systems that can, in principle, be identifiable from the given task dataset. In the setting where tasks are sampled from nearly identical systems, Meta-SysId may not be able to infer context properly. We expect such limitations can be overcome by augmenting the learning model with the proper selection of priors as similarly done in various inverse problem domains.

References

[1] L. Ljung, "Perspectives on system identification," *Annual Reviews in Control*, vol. 34, no. 1, pp. 1–12, 2010.

[2] E. Schrödinger, "An undulatory theory of the mechanics of atoms and molecules," *Physical Review*, vol. 28, no. 6, p. 1049, 1926.

[3] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *Journal of Political Economy*, vol. 81, no. 3, pp. 637–654, 1973.

[4] S. W. Hawking, "Particle creation by black holes," in *Euclidean quantum gravity*, World Scientific, 1975, pp. 167–188.

[5] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[6] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[7] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas, *et al.*, "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature*, vol. 602, no. 7897, pp. 414–419, 2022.

[8] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer, "Machine learning–accelerated computational fluid dynamics," *Proceedings of the National Academy of Sciences*, vol. 118, no. 21, 2021.

[9] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *arXiv preprint arXiv:2004.05439*, 2020.

[10] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning (ICML)*, 2017.

[11] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *arXiv preprint arXiv:1803.02999*, 2018.

[12] L. Bertinetto, J. F. Henriques, P. H. Torr, and A. Vedaldi, "Meta-learning with differentiable closed-form solvers," in *International Conference on Learning Representations (ICLR)*, 2019.

[13] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine, "Meta-learning with implicit gradients," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[14] K. Lee, S. Maji, A. Ravichandran, and S. Soatto, "Meta-learning with differentiable convex optimization," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[15] M. Blondel, Q. Berthet, M. Cuturi, R. Frostig, S. Hoyer, F. Llinares-López, F. Pedregosa, and J.-P. Vert, "Efficient and modular implicit differentiation," *arXiv preprint arXiv:2105.15183*, 2021.

[16] R. Liao, Y. Xiong, E. Fetaya, L. Zhang, K. Yoon, X. Pitkow, R. Urtasun, and R. Zemel, "Reviving and improving recurrent back-propagation," in *International Conference on Machine Learning (ICML)*, 2018.

[17] A. Antoniou, H. Edwards, and A. Storkey, "How to train your MAML," in *International Conference on Learning Representations (ICLR)*, 2019.

[18] H.-J. Ye and W.-L. Chao, "How to train your MAML to excel in few-shot classification," in *International Conference on Learning Representations (ICLR)*, 2022.

[19] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *International Conference on Machine Learning (ICML)*, 2016.

[20] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," in *International Conference on Learning Representations (ICLR)*, 2018.

[21] M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende, and S. A. Eslami, "Conditional neural processes," in *International Conference on Machine Learning (ICML)*, 2018.

[22] J. Gordon, W. P. Bruinsma, A. Y. Foong, J. Requeima, Y. Dubois, and R. E. Turner, "Convolutional conditional neural processes," in *International Conference on Learning Representations (ICLR)*, 2020.

[23] H. Kim, A. Mnih, J. Schwarz, M. Garnelo, A. Eslami, D. Rosenbaum, O. Vinyals, and Y. W. Teh, "Attentive neural processes," in *International Conference on Learning Representations (ICLR)*, 2019.

[24] B. Amos, "Tutorial on amortized optimization for learning to optimize over continuous domains," *arXiv preprint arXiv:2202.00665*, 2022.

[25] G. Scutari, D. P. Palomar, F. Facchinei, and J.-S. Pang, "Monotone games for cognitive radio systems," in *Distributed Decision Making and Control*, Springer, 2012, pp. 83–112.

[26] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, "Learning to simulate complex physics with graph networks," in *International Conference on Machine Learning (ICML)*, 2020.

[27] J. Godwin, M. Schaarschmidt, A. L. Gaunt, A. Sanchez-Gonzalez, Y. Rubanova, P. Veličković, J. Kirkpatrick, and P. Battaglia, "Simple GNN regularisation for 3d molecular property prediction and beyond," in *International Conference on Learning Representations (ICLR)*, 2022.

[28] J. Brandstetter, D. E. Worrall, and M. Welling, "Message passing neural PDE solvers," in *International Conference on Learning Representations (ICLR)*, 2021.

[29] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," in *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014.

[30] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer science & business media, 2013.

[31] F. Allgöwer and A. Zheng, *Nonlinear model predictive control*. Birkhäuser, 2012, vol. 26.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[33] S. M. Richards, N. Azizan, J.-J. Slotine, and M. Pavone, "Adaptive-control-oriented meta-learning for nonlinear systems," in *Robotics: Science and Systems*, 2021.

[34] E. Branlard, "Wind energy: On the statistics of gusts and their propagation through a wind farm," *ECN Wind Memo*, vol. 7, pp. 5–9, 2009.

[35] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[36] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[37] B. Yoo, J. Lee, J. Ju, S. Chung, S. Kim, and J. Choi, "Conditional temporal neural processes with covariance loss," in *International Conference on Machine Learning (ICML)*, 2021.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[39] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[40] S. M. Richards, N. Azizan, J.-J. Slotine, and M. Pavone, "Control-oriented meta-learning," *arXiv preprint arXiv:2204.06716*, 2022.

[41] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 11, 2008.

[42] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[43] M. Poli, S. Massaroli, A. Yamashita, H. Asama, J. Park, and S. Ermon, "TorchDyn: Implicit models and neural numerical methods in PyTorch,"

A Experiment details

A.1 Polynomial regression

In this section, we provide the model architectures, training process, and additional experimental results for the polynomial regression task.

Model architecture We employ `MAML` [10] as an optimization-based meta-learning baseline and a self-attention model `Attn` [38] as a black-box meta-learning baseline. All models use the same multi-layer perceptron (MLP) for f_θ and are trained with the same data batches for fair comparisons.

For brevity, we refer to a multi-layer perceptron (MLP) with hidden dimensions $n_1, n_2, ... n_l$ for each layer and hidden activation `act`, as $\text{MLP}(n_1, n_2, ..., n_l; \text{act})$. We refer to a cross multi-head attention block [38] with h heads and x hidden dimensions as $\text{X.MHA}(h \times x)$.

Table A.1: Polynomial regression model architectures

	Context encoder	f_θ	Inner step K	Inner step size α	τ
Meta-SysId	−	MLP(1+32, 64, 32, 1;SiLU [39])	100	0.001	0.1
MAML	−	MLP(1, 64, 32, 1;SiLU)	4	0.001	−
Attn	MLP(2,32)/MLP(1,32)-X.MHA(4×32)	MLP(1+32, 64, 32, 1;SiLU)	−	−	−

Table A.1 summarizes the network architectures. The green colored values indicate the dimension of context c. For MAML, we also perform hyperparameter search to optimize K. We found that `MAML` with $K > 5$ underperforms, as compared to $K = 5$.

Training details We train all models with mini-batches of 256 polynomials for 4,048 epochs using Adam [32] with a fixed learning rate of 0.001.

Table A.2: Polynomial regression MSE with increasing context points N.

N	Meta-SysId	MAML	Attn	System. ID
1	**0.1630**	0.6438	0.3905	0.4989
3	**0.0523**	0.0815	0.0826	0.0840
5	**0.0268**	0.0473	0.0495	0.0187
10	**0.0097**	0.0260	0.0204	0.0000

Additional experiments We evaluate the predictive performances of the models with different values of N on the test dataset. Table A.2 shows the average mean squared errors (MSE) of the meta-learning methods and the classical system identification that knows the exact functional form and solve the system identification problem with the context points (`System Id`). As shown in Table A.2, Meta-SysId outperforms all meta-learning baselines for all N. Noticeably, it even shows better predictions than `System Id` when $N < 5$.
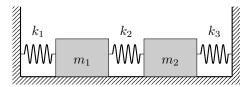
A.2 Time-series prediction

Figure A.1: Target mass-spring system. The models require to adapt to the change of spring constants (k_1, k_2, k_3) and masses (m_1, m_2).

Mass-spring system For the time-series prediction task, we consider a frictionless three-spring two-mass system shown in Fig. A.1 with mass positions x_1 and x_2 governed by the following dynamics:

$$\begin{pmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{pmatrix} = \underbrace{\begin{bmatrix} -\frac{k_1+k_2}{m_1} & \frac{k_2}{m_1} \\ \frac{k_2}{m_2} & -\frac{k_2+k_3}{m_2} \end{bmatrix}}_{K} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \tag{A.1}$$

where K is a coefficient matrix with spring constants (k_1, k_2, k_3) and masses (m_1, m_2).

Model architecture For f_θ, we employ the 1D-CNN model from [28], which stacks a MLP, 1D-CNN, and consistency decoder [28]. For brevity, we refer to a 1D-CNN layer with x input channels, y output channels, and filter size w as `Conv(x,y,w)`, a consistency decoder as `C.Dec`, and the bi-directional GRU [29] with hidden dimension x as `GRU(x)`.

Table A.3: Mass-spring prediction model architectures

	Context encoder	f_θ
Meta-SysId	—	MLP(4×25+64, 64, 32, 4×25;SiLU)-Conv(4, 4, 8)-SiLU-Conv(4,4,1)-C.Dec
MAML	—	MLP(4×25, 64, 32, 4×25;SiLU)-Conv(4, 4, 8)-SiLU-Conv(4,4,1)-C.Dec
RNN	GRU(64)-MLP(64, 64)	MLP(4×25+64, 64, 32, 4×25;SiLU)-Conv(4, 4, 8)-SiLU-Conv(4,4,1)-C.Dec
NoAdapt	—	MLP(4×25, 64, 32, 4×25;SiLU)-Conv(4, 4, 8)-SiLU-Conv(4,4,1)-C.Dec

Table A.3 summarizes the network architectures. The green colored values indicate the dimension of context c. We set the inner step K and step size α as 50/5 and 0.001/0.001 for `Meta-SysId` and `GrBAL`, respectively, and τ as 0.1.

Training details We train all models with mini-batches of size 512 for 512 epochs using Adam [32] with a fixed learning rate of 0.001 and pushforward regularization [28]. For the meta-learning models (i.e., `Meta-SysId`, `MAML`, `ReBAL`), we use the past observations from the previous $2N$ to N steps as the input of the adaptation process.

A.3 Model-based control

A.3.1 Model predictive control formulation

We first give a brief introduction of the control algorithm we use. Also known as receding horizon control, model predictive control (MPC) is a class of flexible control algorithms capable of taking into consideration constraints and nonlinearities [30], [31]. MPC interleaves actions with planning over finite time windows which are then shifted forward in a *receding* manner. The control problem is then solved for each window by *predicting* future trajectories with a candidate controller u and then adjusting the controller to optimize the cost function J. The first control action is taken and the optimization is repeated *online* until the end of the episode.

A.3.2 System dynamics

We consider a Planar Fully Actuated Rotorcraft (PFAR) [33], [40] with position x, height y, angle ϕ, and dynamics governed by:

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\phi} \end{pmatrix} = \underbrace{\begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R(\phi)} \begin{pmatrix} u_x - \beta_1 v_1|v_1| \\ u_y - \beta_2 v_2|v_2| \\ u_\phi \end{pmatrix} + \begin{pmatrix} 0 \\ -g \\ 0 \end{pmatrix}, \tag{A.2}$$
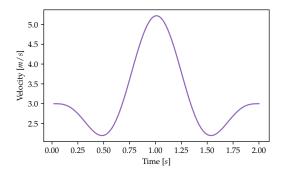
Figure A.2: Extreme Operating Gust (EOG) profile used in the variable context experiment. This type of wind gust is modeled by a characteristic "Mexican hat" shape.

where $R(\phi)$ is a rotation matrix, $u = (u_x, u_y, u_\phi)$ are normalized control actions on their respective dimensions (i.e. thrusts and a torque), $g = 9.81 \ m/s^2$ is the gravitational acceleration, and β_1 and β_2 are drag coefficients which we set to 0.1 and 1 respectively. Velocities v_1 and v_2 can be described by:

$$
\begin{aligned}
v_1 &= (\dot{x} - w)\cos\phi + \dot{y}\sin\phi, \\
v_2 &= -(\dot{x} - w)\sin\phi + \dot{y}\cos\phi,
\end{aligned}
\tag{A.3}
$$

where w is wind acting on the x direction.

A.3.3 Data generation details

We generate a total of 500, 10 seconds–long, reference trajectories by fitting a smooth polynomial spline trajectory on a random walk of the state parameters and simulating the system with a PD controller with proportional and differential gain $k_p = 10$, $k_d = 0.1$ respectively, and trajectory–wise constant wind on the x-axis drawn from $\mathcal{U}(0, 8) \ m/s$. We note that the PD–controlled trajectories are not "optimal" as the tracking performance is not essential; we aim only at obtaining a rough distribution of state-control sequences to train the models.

A.3.4 Model training

We train Meta-SysId and the RNN for 512 epochs with the Adam optimizer [32] with learning rate of 10^{-3}. Context inference in Meta-SysId is performed by iterating for 50 steps, also with the Adam optimizer and a learning rate of 10^{-3}. the latent context dimension is set to 32. Models are trained on the dataset described in Appendix A.3.3 with the sub-sampled training trajectories. Each sub-trajectory comprises 25-time steps of historical observations and 25-time steps of future observation. All models utilize historical observations and future observations for context inference and meta prediction.

We also tested the optimization-based meta-learning approach (i.e., MAML) to the specified training scheme. However, using our computing infrastructures, training of the optimization-based meta-learning approach took more than an hour for a single epoch. As all models are trained for 512 epochs, we exclude the MAML-based approach from the baseline for fair comparisons.

A.3.5 Additional experimental details

Drone trajectory tracking After generating reference trajectories, we employ MPC to obtain control actions over a receding horizon of 10 time steps. The cost function J is chosen as the mean squared error between each target and real position over time.

Drone stabilization under variable wind profiles We also consider stabilizing the system to the origin, i.e. $q^* = (x, y, \phi, \dot{x}, \dot{y}, \dot{\phi}) = (0, 0, 0, 0, 0, 0)$, under Extreme Operating Gusts (EOG). EOGs
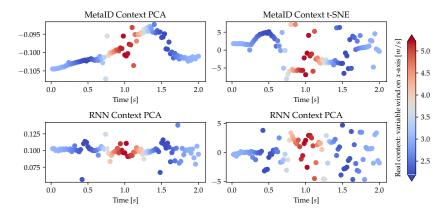
Figure A.3: PCA and t-SNE visualizations of Meta-SysId and the RNN baseline. Even though the latent context is 32–dimensional, Meta-SysId can reconstruct the roughly smooth dynamics of the wind gust.

can be characterized by a decrease in wind speed, followed by a steep rise, subsequent drop, and final rise back to the original average wind speed [34]:

$$w(t) = \begin{cases} \bar{W} - 0.37 W_{gust} \sin\left(\frac{3\pi t}{T}\right)(1 - \cos\left(\frac{2\pi t}{T}\right)) & \text{for } t_0 \leq t \leq T \\ \bar{W} & \text{otherwise} \end{cases} \tag{A.4}$$

where $\bar{W}$ is the average wind speed and W_{gust} is the wind gust magnitude. Figure A.2 shows the wind gust profile used in the drone stabilization experiment. We employ MPC with a receding horizon of 10 time steps whose cost function J is the mean squared error between each position in the trajectory and q^*.

Latent context visualizations   We employ PCA [35] and t-SNE [41] to visualize the latent context. In particular, we employ the 1–dimensional version in which the latent context of size 32 is mapped to a value in $\mathbb{R}$. We visualize the operation for each point in time and in Fig. A.3. We observe that in `Meta-SysId` the latent context retains more *smoothness* compared to RNN. We speculate that this can at least partially explain why our proposed method yields better performance in control settings where fast online changes occur.

A.4   Traffic flow prediction

In this section, we detail the model architectures and training process for the traffic flow prediction task.

Model architecture   For the traffic flow prediction `Meta-SysId` model, we employ the same model architecture of STGCN(Cheb) [36] for f_θ, but modify the input dimension from 15 (i.e., 60-minute observations) to 15+32 to accommodate the context as an extra input.

Training details   We train `Meta-SysId` with the same hyperparameters of STGCN(Cheb) [36]. To find context, we use 10 inner optimization steps and a step size α of 0.001.

A.5   Hardware and Software

Experiments were carried out on a machine equipped with an AMD THREADRIPPER 2990WX CPU with 64 threads and an NVIDIA RTX 3090 graphic card. Software–wise, we used `PyTorch` [42] for deep learning and the `torchdyn` [43] library for efficient ODE solvers.
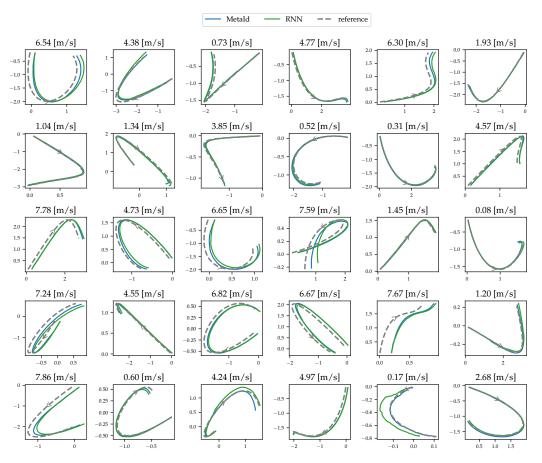
15

Figure A.4: Comparisons between reference and controlled trajectories.

B Additional results

B.1 Model-based control results

Fig. A.4 visualizes the complete reference tracking results comparing Meta-SysId with the RNN baseline.