GENERALIZING TO NEW DYNAMICAL SYSTEMS THROUGH FIRST-ORDER CONTEXT-BASED ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we propose FOCA (First-Order Context-based Adaptation), a learning framework to model sets of systems governed by common but unknown laws that differentiate themselves by their context. Inspired by classical modeling-and-identification approaches, FOCA learns to represent the common law through shared parameters and relies on online optimization to compute system-specific context. Due to the online optimization-based context inference, the training of FOCA involves a bi-level optimization problem. To train FOCA efficiently, we utilize an exponential moving average (EMA)-based method that allows for fast training using only first-order derivatives. We test FOCA on polynomial regression and time-series prediction tasks composed of three ODEs and one PDE, empirically finding it outperforms baselines.

1 INTRODUCTION

Scientists and engineers have made tremendous progress on modeling the behavior of natural and engineering systems and optimizing model parameters to best describe the target system (Ljung, 2010). This *modeling and system identification* paradigm has made remarkable advances in modern science and engineering (Schrödinger, 1926; Black & Scholes, 1973; Hawking, 1975). However, applying this paradigm to complex systems is difficult because the mathematical modeling of systems requires a considerable degree of domain expertise, and finding the best system parameters requires massive experimentation.

The availability of large datasets and advances in deep learning tools have made it possible to model a target system without specific mathematical models, relying instead on flexible model classes (Brunton et al., 2016; Gupta et al., 2020; Menda et al., 2020; Jumper et al., 2021; Kochkov et al., 2021; Degrave et al., 2022). However, when the characteristics of target systems change (e.g., system parameters, boundary conditions), the flexibility of data-driven models makes them difficult to adapt.

Deep learning approaches typically handle the contextual change by collecting data from the new behavioral mode and re-training the model on the new dataset. However, this approach can be impractical, especially when the system is complex and context change is frequent. We are interested in developing a framework that learns a common shared model of the systems and inferring the context that best describes the target system to predict response. Our study considers a target system whose input x and response y can be described by $y = f(x, c)$, where f denotes the function class shared by the target systems and c denotes the system-specific context.

One possible approach for modeling such target systems is meta-learning (Hospedales et al., 2021), which learns how to adapt to new systems. Meta-learning is typically a combination of an adaptive mechanism and training for the adaptation. One typical meta-learning approach is to use an encoder that takes the adaptation data and returns the learned context (Santoro et al., 2016; Mishra et al., 2018; Garnelo et al., 2018; Kim et al., 2019). Although encoder-based adaptation schemes require constant memory usage, their parameterized encoders limit the adaptation capability. Other approaches (pre) train the parameters on the dataset collected from the various modes and update all parameters using gradient-descent (Finn et al., 2017; Nagabandi et al., 2018; Rajeswaran et al., 2019). Despite their effective adaptability, those approaches are often prone to (meta) over-fitting (Antoniou et al., 2019), especially when the adaptation target is complex and adaptation data is scarce. Instead of updating all parameters, Raghu et al. (2019); Zintgraf et al. (2019) update a subset

Table 1: Comparison of context-based generalization approaches. The memory column specifies the additional memory requirements for adaptation during the training phase of the algorithms. $|\cdot|$ denotes the number of elements of $\cdot$.

	Training models (parameters)	Adaptation mechanism	Memory
`Encoder`	f_θ, g_ϕ	$\hat{c} = g_\phi(\mathcal{D})$	$\mathcal{O}(\lvert\phi\rvert)$
`CAVIA`	f_θ	$\hat{c}_{k+1} = \hat{c}_k - \lambda \nabla_{\hat{c}_k} \mathcal{L} \sum_{(x,y)\in\mathcal{D}} (f_\theta(x, \hat{c}_k), y)$, $\hat{c}_0 = 0$, $\hat{c} = c_K$, where K is the adaptation steps.	$\mathcal{O}(\lvert\theta\rvert \cdot K)$
`CoDA`	$f_\theta, \boldsymbol{W}$	$\hat{c} = \arg\min_c \sum_{(x,y)\in\mathcal{D}} \mathcal{L}\left(f_{\theta+\boldsymbol{W}c}(x), y\right)$	$\mathcal{O}(\lvert\boldsymbol{W}\rvert)$
`FOCA`	f_θ	$\hat{c} = \arg\min_c \sum_{(x,y)\in\mathcal{D}} \mathcal{L}\left(f_{\bar{\theta}}(x, c), y\right)$, where $f_{\bar{\theta}}$ is an EMA copy of f_θ.	$\mathcal{O}(1)$

of parameters, which we call context $\hat{c}$, while fixing the remainder. Although this modeling approach is effective, the training requires computing higher-order derivatives.

Many training method for meta-learning have been proposed (Finn et al., 2017; Nichol et al., 2018; Rajeswaran et al., 2019; Deleu et al., 2021). Typically encoder-based meta-learning trains an encoder and a prediction model jointly and no online optimization-based adaptation occurs. The training of gradient-based meta-learning is typically cast as bi-level optimization. Typical gradient-based meta-learning is often carried out by propagating through the update steps, which requires higher-order derivative calculations. To avoid such issues, Finn et al. (2017); Nichol et al. (2018) propose a first-order approximation of derivatives to update the (meta) parameters. Alternatively, the implicit gradient method can be used to lower the computational burden (Rajeswaran et al., 2019), but gradient computation errors can be significant and result in performance degradation (Liao et al., 2018; Zhou et al., 2019; Blondel et al., 2021; Chen et al., 2022).

In this paper, we propose FOCA (First-Order Context-based Adaptation), a context-based meta-learning method that is specialized for complex dynamical systems whose behavior can be characterized by a common mathematical model f and a context c. Specifically, FOCA considers target systems of the form $y = f_\theta(x, \hat{c})$, where f_θ denotes the learned function class shared by the target systems and $\hat{c}$ denotes the inferred system-specific context. FOCA learns the function class f_θ during training and solves the system identification problem through numerical optimization to find the proper $\hat{c}$. The online context optimization bypasses the limitation of encoder-based approaches, but it entails a higher computational burden. To train FOCA efficiently, we thus propose using an exponential moving average (EMA) based training method, which operates with first-order derivatives and no additional memory usage. From our experiments, we also confirm that EMA-based training decreases the computational burden of training and improves the generalization performance of FOCA. The contributions of this work are summarized as follows:

- We propose FOCA, a learning framework specialized for modeling complex dynamical system families that are described by an unknown common law and system-specific context.
- We propose an EMA-based training method for training FOCA that overcomes the burden of second-order derivative calculations while showing better generalization results compared to other training methods.
- We empirically demonstrate that FOCA outperforms or is competitive to various meta-learning baselines in static function regression and time-series prediction tasks evaluated both in-distribution and out-of-distribution.

2 Related work

Learning to generalize to the new systems. Transfer learning (Zhuang et al., 2020) attempts to generalize learned models to new systems (or tasks) by fine-tuning a small number of parameters to the new tasks. As a more direct way of generalization to new systems, meta-learning learns how to adapt quickly to new systems. In particular, gradient-based meta-learning (GBML) approaches perform few-step gradient descent updates of the model parameter θ for adaptation (Finn et al.,

2017; Nichol et al., 2018). Focusing on the empirical evidence that the plain GBML is prone to overfit the training tasks, CAVIA (Zintgraf et al., 2019) performs gradient-based updates for a small dimensional context vector c. On the other hand, encoder-based meta-learning methods jointly train an encoder network g_ϕ alongside the prediction model f_θ (Santoro et al., 2016; Mishra et al., 2018; Garnelo et al., 2018; Kim et al., 2019; Gordon et al., 2020).

A few works investigate how to solve our target problem (i.e., learning a common unknown rule f of static or dynamic systems with the learned adaptation mechanism). Lee et al. (2020b) applies GBML to generalize over Hamiltonian systems. LEADS (Yin et al., 2021) jointly learns the system specific networks g_ϕ and shared network f_θ, and predicts the response of the dynamic system with $f_\theta + g_\phi$. CoDA (Kirchmeyer et al., 2022) learns the common model parameters θ, a linear basis $\boldsymbol{W}$, and context vectors $\hat{c}$ while defining $f_\theta(\cdot, \hat{c})$ as $f_{\theta+\boldsymbol{W}\hat{c}}(\cdot)$. To adapt to the new system with the observation dataset $\mathcal{D}$, it solves an optimization, $\min_c \sum_{(x,y)\in\mathcal{D}} \mathcal{L}\left(f_{\theta+\boldsymbol{W}c}(x), y\right)$. In contrast, FOCA adapts by solving Eq. (6), which is an effective method to generalize to systems that do not require any structural assumptions. This approach is different from LEADS, where the adaptation is sum-decomposable, and from CoDA, where the adaptive parameters are a span of a linear basis. We summarize the differences between FOCA and baselines in Table 1.

Solving bi-level optimization to learn. Training FOCA involves solving a bi-level optimization. Differentiable optimization layers (Amos & Kolter, 2017; Agrawal et al., 2019; Meng et al., 2021) encapsulate an optimization problem as a layer of neural networks. As they employ optimization as a layer of the network, the training becomes a bi-level optimization problem. The backward path of differentiable optimization layers is often implemented by backpropagating the optimization steps or implicit gradient methods. As a special case, Bertinetto et al. (2019) employs inner optimization problems that can be solved analytically.

In the context of meta-learning, Bertinetto et al. (2019) considers solving the analytically solvable inner optimization problem. However, such approaches limit the flexibility of network parameterization or adaptation mechanisms. As a more flexible approach, Implicit MAML (Rajeswaran et al., 2019) extends MAML to overcome the extensive memory usage of adaptation and allows for more inner adaptation steps without memory issues. However, implicit gradient methods can calculate inexact meta-gradients (Deleu et al., 2021), and, as we show in the experiments, such errors can degrade performance. Instead, we propose to solve the bi-level optimization training problem with the EMA method, which also has constant memory usage. We empirically show that the EMA-based training method outperforms implicit gradient-based methods on test metrics.

3 Preliminaries

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

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

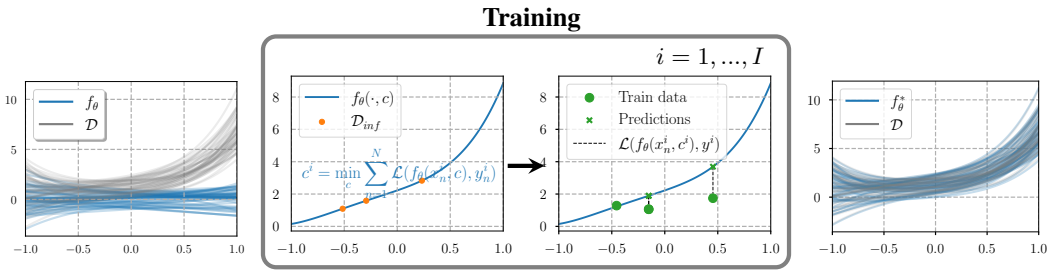
$$y = f(x, c). \tag{1}$$

Classical parameter identification aims at inferring the (optimal) context c^* with data from a task $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$. Under the assumption that (x_n, y_n) pairs in $\mathcal{D}$ are generated from the same system, 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 learning from experiences of related tasks. Assuming that tasks (i.e., the systems of different contexts) are drawn from a distribution $p(\mathcal{T})$ and yield task inference data $\mathcal{D}_{\text{inf}}^{\mathcal{T}}$ and (meta)-training



(a) Initial $f_\theta(\cdot, \cdot)$ (b) Task adaptation (c) Prediction (d) Trained $f_\theta^*(\cdot, \cdot)$

Figure 1: Training of FOCA: (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 context inference 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 $f_\theta^*(\cdot, \cdot)$. FOCA is trained by repeating (b), (c), and minimizing $\mathcal{L}(f_\theta(x_n^i, c^i), y_n^i)$.

data $\mathcal{D}_{\text{tr}}^{\mathcal{T}}$, the meta-learning objective can be defined as follows:

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

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

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 learns θ and ψ to infer a model from $\mathcal{D}_{\text{inf}}^{\mathcal{T}}$ that minimizes the error on $\mathcal{D}_{\text{tr}}^{\mathcal{T}}$. Optimized meta-parameters θ^* and ψ^* can then be used to "quickly" adapt to new tasks from $p(\mathcal{T})$.

4 Methodology

In this section, we present First-Order Context-based Adaptation (FOCA) to learn to jointly identify and predict systems in the same family. We first introduce the training problem of FOCA and then discuss the EMA-based training method for efficiently solving the training problem. We also provide our code as supplementary material.

4.1 Problem formulation

We formulate the learning problem of FOCA as a bi-level optimization where the inner optimization Eq. (6) is for inferring the context, and the outer optimization Eq. (5) is for learning f_θ conditioned on the identified context. The proposed optimization problem is as follows:

$$\min_\theta \sum_{\mathcal{D}^i \in \mathcal{D}} \sum_{(x_n^i, y_n^i) \in \mathcal{D}_{\text{tr}}^i} \mathcal{L}\left(f_\theta(x_n^i, \hat{c}^i), y_n^i\right) \tag{5}$$

$$\text{subject to} \quad \hat{c}^i = \arg\min_c \sum_{(x_n^i, y_n^i) \in \mathcal{D}_{\text{inf}}^i} \mathcal{L}\left(f_\theta(x_n^i, c), y_n^i\right), \tag{6}$$

where $\mathcal{D}_{\text{inf}}^i$ and $\mathcal{D}_{\text{tr}}^i$ are the context inference and training dataset of the ith system, which satisfy $\mathcal{D}_{\text{inf}}^i \cap \mathcal{D}_{\text{tr}}^i = \emptyset$ and $\mathcal{D}_{\text{inf}}^i \cup \mathcal{D}_{\text{tr}}^i \subset \mathcal{D}^i$, $\mathcal{D} = \{\mathcal{D}^i\}_{i=1,\dots}$ is a collection of datasets from multiple systems, f_θ is a prediction model parameterized by θ, and $\mathcal{L}$ is a loss metric.

The prediction model f_θ is trained by solving Eqs. (5) and (6). The trained f_θ^* describes the common structure of the systems in $\mathcal{D}$. Identification of a specific function (i.e., adaptation) is done by solving Eq. (6) with the data points sampled from the specific system. After finding c^i, we can query for meta-test predictions with $f_\theta(\cdot, c^i)$ as visualized in Fig. 1 (b) and (c).

FOCA can be viewed as a special case of the meta-learning recipe explained in Section 3 by defining $\theta' = [\theta, \hat{c}^i]$ and setting Eq. (4) as Eq. (6). Even though FOCA is formulated similarly to optimization-based meta-learning methods, the optimization variables for the adaptation is $\hat{c}^i$ rather than θ. In terms of implementation, this algorithmic selection allows us to batch-solve inner-level optimization with standard automatic differentiation tools. FOCA can therefore be trained efficiently. We can also interpret this clear separation of θ and c as capturing function classes and their coefficients separately. We further inspect this view of FOCA in Section 5.1.

Algorithm 1: Training FOCA with exponential moving average (EMA)

Input: Prediction model f_θ, Tasks $\mathcal{D}$, inner optimization steps K, inner optimization step size α, Weighting factor τ, context regularization weight λ

$\bar{\theta} \leftarrow \theta$, $f_{\bar{\theta}} \leftarrow f_\theta$ `// Initialize the target model`
for $1, 2, \ldots$ **do**
  Sample batch of tasks $\mathcal{D}^i \sim \mathcal{D}$
  for $\mathcal{D}^i \in \mathcal{D}$ **do**
    construct $\mathcal{D}^i_{\text{inf}}$ and $\mathcal{D}^i_{\text{tr}}$ from $\mathcal{D}^i$
    $\hat{c}^i \leftarrow 0$
    for $k = 1, \ldots, K$ **do**
      $\mathcal{L}(c^i) = \sum_{(x^i_n, y^i_n) \in \mathcal{D}^i_{\text{inf}}} \mathcal{L}(f_{\bar{\theta}}(x^i_n, c^i), y^i_n) + \lambda \|c^i\|$
      $\hat{c}^i \leftarrow \hat{c}^i - \alpha \nabla_{c^i} \mathcal{L}(c^i)$ `// Solve Eq. (6)`
    end
  end
  Evaluate $\mathcal{L}(\theta) = \sum_{\mathcal{D}^i} \sum_{(x^i_n, y^i_n) \in \mathcal{D}^i_{\text{tr}}} \mathcal{L}(f_\theta(x^i_n, \hat{c}^i), y^i_n)$
  $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$
  $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$ `// Update the target model`
end

4.2 Training FOCA

Training FOCA with gradient descent involves computation of the partial derivatives of $\mathcal{L}^{\text{upper}} = \sum_{\mathcal{D}^i \in \mathcal{D}} \sum_{(x^i_n, y^i_n) \in \mathcal{D}^i_{\text{tr}}} \mathcal{L}\left(f_\theta(x^i_n, \hat{c}^i), y^i_n\right)$ with respect to θ. We express $\partial \mathcal{L}^{\text{upper}} / \partial \theta$ using the chain rule taking $\hat{c}^i$ as the intermediate variable,

$$\frac{\partial \mathcal{L}^{\text{upper}}}{\partial \theta} = \frac{\partial \mathcal{L}^{\text{upper}}}{\partial \hat{c}^i} \frac{\partial \hat{c}^i}{\partial \theta}. \tag{7}$$

Here, $\partial \mathcal{L}^{\text{upper}} / \partial \hat{c}^i$ can be computed via an automatic differentiation package. However, the computation of $\partial \hat{c}^i / \partial \theta$ is less straightforward because of the implicit relations between $\hat{c}^i$ and θ. That is, their relations are defined through an optimization problem rather than a computational chain.

One possible approach to calculate $\partial \hat{c}^i / \partial \theta$ is to backpropagate through the inner optimization steps, which calculates the derivatives by unrolling the inner optimization steps through the computational graph. However, this method is not memory efficient, especially with a large number of model parameters $|\theta|$ or with many inner optimization steps. Another method is to approximate $\partial \hat{c}^i / \partial \theta$ with first-order methods as similarly done in Finn et al. (2017); Nichol et al. (2018). However, as reported by others (Zhou et al., 2019; Jayathilaka, 2019; Chen et al., 2022), this approach often requires techniques to mitigate the approximation errors. Another method calculates $\partial \hat{c}^i / \partial \theta$ using implicit gradients and can bypass the memory usage of the first approach. However, calculating the gradient without error requires solving the inner optimization problem optimally (Liao et al., 2018; Blondel et al., 2021). In practice, this restricts the expressivity of f_θ to architectures that can be solved optimally Eq. (6). We propose an efficient approach to train FOCA that uses no additional memory and allows for flexible architecture of f_θ .

EMA-based training method. Inspired by recent developments in self-supervised learning (Grill et al., 2020; Caron et al., 2021) and the target-network of reinforcement learning (Lillicrap et al., 2016), we use a simple method based on the exponential moving average (EMA) to train FOCA. The core idea is to consider $\hat{c}$ as an independent input given from outside of the gradient chain. First, we copy θ to create a delayed target model $f_{\bar{\theta}}$ parameterized by $\bar{\theta}$. We use $f_{\bar{\theta}}$ to infer $\hat{c}^i$ with gradient descent on Eq. (6). We then update θ by using $\nabla_\theta \mathcal{L}(\theta) = \partial \mathcal{L}^{\text{upper}} / \partial \theta$ evaluated on $\hat{c}^i$. As the parameters of f_θ and $f_{\bar{\theta}}$ are separate, this avoids differentiating the argmin operator to calculate $\nabla_\theta \mathcal{L}(\theta)$. After updating θ, we update θ as $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$ with $0 \leq \tau \ll 1$. The training procedure is summarized in Algorithm 1.

The proposed training algorithm utilizes $f_{\bar{\theta}}$ (i.e., the delayed copy of f_θ) to infer $\hat{c}^i$ by numerically solving Eq. (6). When we train f_θ, it takes $\hat{c}^i$ as an input that contains meaningful information that

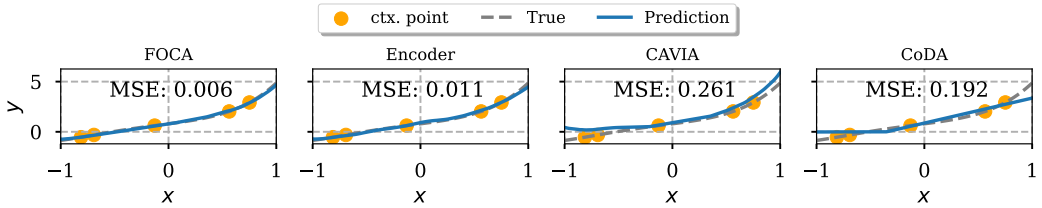


Figure 2: Polynomial regression results. All models infer the polynomial from the given context points (●). The prediction results and ground truth values are visualized with the blue-solid (—) and gray-dashed (- -) lines, respectively.

differentiates the system i from the other systems. At the early training phase, $\hat{c}^i$ might have no meaning (i.e., the different $\hat{c}^i$s do not change the prediction results). However, this phenomenon is resolved during training because f_θ is trained as if $\hat{c}^i$ is a (learned) system-differentiating context. Furthermore, disentangling the model for prediction and inference results in a practical advantage: we can employ any optimization algorithm (even non-differentiable ones) to solve Eq. (6). The proposed method shows its advantages in memory usage and implementation.

We also observed that EMA-based training can increase the generalization performance over the other training methods. We discuss this aspect with the experimental results in Section 5.3.

5 Experiments

In this section, we discuss experiments that show the properties of FOCA with static regression problems (Section 5.1), demonstrate its effectiveness in predicting the response of dynamics systems (Section 5.2), and investigate the proposed EMA-based training method (Section 5.3).

Baselines. We compare FOCA to three baselines that adapt to the tasks by inferring $\hat{c}$. The first two baselines are representative methods that find $\hat{c}^i$ via the inference schemes discussed in Section 1. The last baseline, `CoDA` (Kirchmeyer et al., 2022), is our closest baseline in terms of application: a meta-learning method tailored for generalization of physical systems. To make fair comparisons, all models use the same network architecture $f_\theta(\cdot)$ for each task. The baselines are as follows:

- `Encoder` employs an encoder network to extract the context $\hat{c}^i$ from $\mathcal{D}^i_{\text{inf}}$. That is, $\hat{c}^i = g_\phi(\mathcal{D}^i_{\text{inf}})$, where g_ϕ is the encoder network. We employ SNAIL (Mishra et al., 2018) and CaDM (Lee et al., 2020a)-based g_ϕ for static and time-series tasks, respectively.
- `CAVIA` (Zintgraf et al., 2019) performs a few gradient steps on Eq. (6) to extract the context $\hat{c}^i$ from $\mathcal{D}^i_{\text{inf}}$ and train f_θ via the backpropagation-through-optimization-step scheme.
- `CoDA` (Kirchmeyer et al., 2022) learns the common model parameters θ, a linear basis $\boldsymbol{W}$, and context vectors $\hat{c}^i$ while defining $f_\theta(\cdot, \hat{c}^i)$ as $f_{\theta+\boldsymbol{W}\hat{c}^i}(\cdot)$. For adaptation to a new system, it solves the optimization problem $\min_c \sum_{(x^i_n, y^i_n)\in\mathcal{D}^i_{\text{inf}}} \mathcal{L}\left(f_{\theta+\boldsymbol{W}c^i}(x^i_n), y^i_n\right)$.

5.1 Static Regression

We first examine polynomial function regression to understand the properties of FOCA. In this task, the objective is to identify a polynomial and make predictions from the identified functions as shown in Fig. 2. We prepare 4th order polynomial datasets by sampling the coefficients from $\mathcal{U}(0.1, 2.5)$, and set $|\mathcal{D}^i_{\text{inf}}|$ and $|\mathcal{D}^i_{\text{tr}}|$ as 5 and 15, respectively. Fig. 3 shows the average of the test root mean squared error (RMSE) values of models. As shown in Fig. 3, FOCA adapts to the different polynomials better than the baseline models. We provide the details of the model architectures and training scheme in Appendix A. We then investigate the properties of FOCA by using the polynomial dataset.

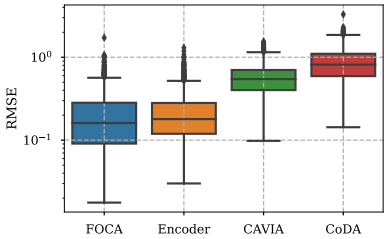


Figure 3: Static regression test RMSE.

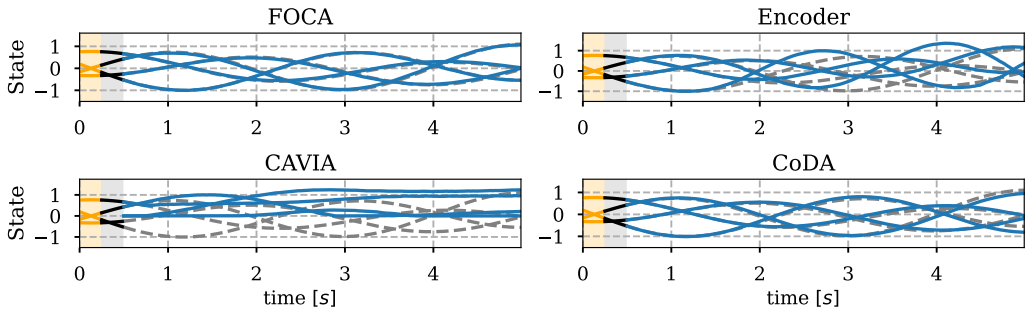


Figure 5: MS prediction results. All models infer the physical constants from historical (shaded in yellow) and current (shaded in gray) state observations, and then predict rollouts of future state trajectories (colored in blue). The gray dashed lines visualize prediction targets. FOCA successfully identifies the target systems and predicts long-term futures.

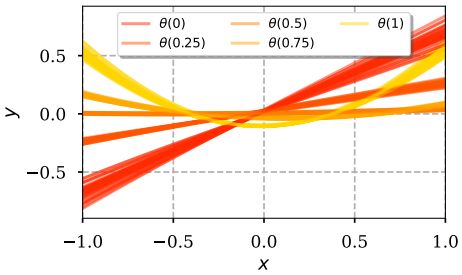


Figure 4: Role of $\hat{c}$ within model classes $\theta(\lambda)$ that interpolate between quadratic and linear.

Do θ and c contain function class and parameter information, respectively? The roles of θ and c are to capture a function class and to identify 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 FOCA 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 $0 \leq \lambda \leq 1$. Fig. 4 shows 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 linear functions when $\lambda = 0$, quadratic functions when $\lambda = 1$, and functions that interpolate between 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).

5.2 Time-series prediction

We evaluate FOCA on time-series prediction in three ODE datasets, including mass-spring systems (MS), Lotka-Volterra equations (LV), glycolytic oscillators (GO), and one PDE dataset derived from the Navier-Stokes equations (NS). In this setting, the objective is first to identify target systems from historical and current state observations and then to predict rollouts of future state trajectories as shown in Fig. 5. We provide details for dataset generation, model architectures, and training schemes in Appendices B to E.

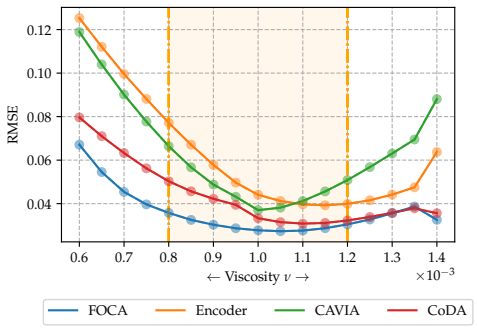


Figure 6: RMSE for different models on the Navier-Stokes equations at different viscosities ν. The orange area denotes the training region.

Task generalization. We study the generalization performances of the models on systems whose context parameters are generated from inside as well as outside the training distribution (i.e., in-training and out-of-training distribution adaptation tasks). We evaluate the models by measuring the average and standard deviation of the rollout prediction root mean squared errors (RMSE) over the test systems. As shown in Table 2, FOCA generally outperforms the baseline algorithms, indicating that FOCA adapts well to new systems regardless of the parameters of the system. To further analyze the performance variation of models to the change of system parameters, we visualize the prediction

Table 2: In/out-of training distribution generalization results. Smaller is better (↓). We measure the RMSE values per trajectory. Average RMSEs and their standard deviations are reported. Best in **bold**; second underlined.

	MS		*LV*		*GO*		*NS*	
	In-training	Out-of training	In-training	Out-of training	In-training	Out-of training	In-training	Out-of training
`Encoder`	2.272 ± 1.442	2.379 ± 1.506	0.137 ± 0.093	0.326 ± 0.362	0.062 ± 0.092	0.156 ± 0.174	0.057 ± 0.016	0.114 ± 0.026
`CAVIA`	1.743 ± 0.595	1.793 ± 0.628	0.312 ± 0.316	0.706 ± 0.551	0.046 ± 0.061	0.157 ± 0.161	0.057 ± 0.011	0.114 ± 0.028
`CoDA`	**0.250** ± 0.201	0.545 ± 0.402	0.258 ± 0.199	0.642 ± 0.349	0.072 ± 0.115	0.269 ± 0.258	0.051 ± 0.034	0.095 ± 0.046
`FOCA`	0.258 ± 0.156	**0.478** ± 0.295	**0.079** ± 0.061	**0.303** ± 0.291	**0.043** ± 0.066	**0.147** ± 0.168	**0.042** ± 0.026	**0.070** ± 0.038

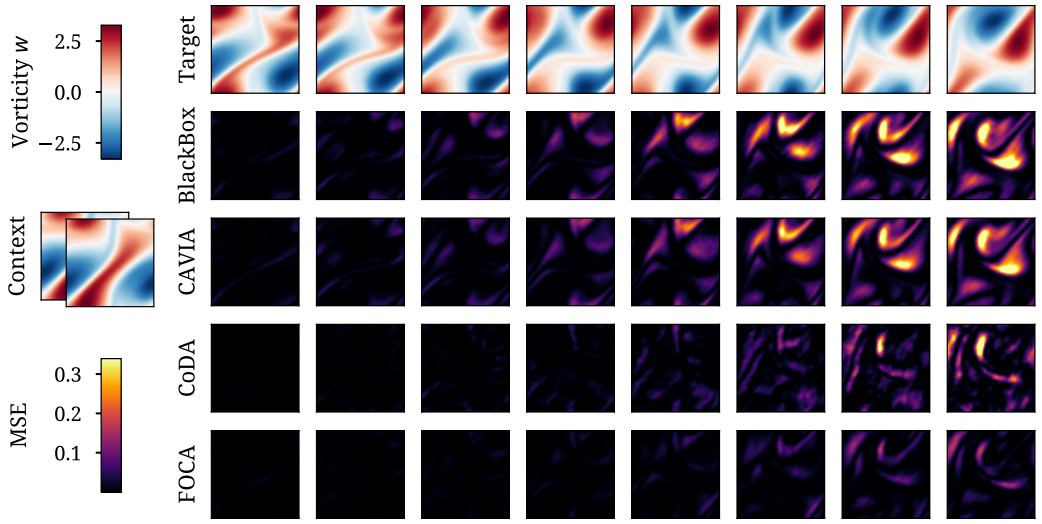


Figure 7: Target solutions and RMSE propagation on the Navier-Stokes equations for different models. FOCA manages to better contain errors even after long autoregressive rollouts and out-of-distribution context samples.

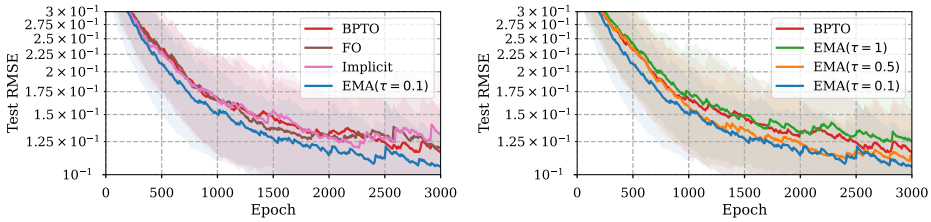
errors of the models on the Navier-Stokes equations for different viscosity values in Fig. 6, while in Fig. 7 we show a sample of target predictions and error propagation for out-of-distribution viscosity $\nu = 10^{-4}$. FOCA shows better prediction results than baselines, not only for new systems with in-distribution contexts but also for out-of-distribution ones. We provide the additional studies for MS, LV, and GO in Appendices B to D, including out-of-distribution error comparison visualizations.

5.3 Training ablation study

In this section, we perform ablation studies to understand the effect of the proposed training scheme on the performance of FOCA.

Is the EMA target network scheme effective? To train FOCA, we employ the delayed target network $f_{\hat{\theta}}$ to infer $\hat{c}^i$, which we consider an independent input when we train f_θ (`EMA`). However, we can consider employing other training methods as mentioned in Section 4.2. We compare against training FOCA by calculating gradients by backpropagating through the unrolled inner optimization steps (`BPTO`), approximating the inner loop gradient with a first-order approximation (`FO`), or employing the implicit method (e.g., Amos & Kolter 2017) for calculating the gradient of the inner optimization (`Implicit`). We compare the test set performance of each method on the LV dataset while applying the same hyperparameters. Fig. 8a visualizes the test performances of each method over the training steps.

As shown in Fig. 8a, `EMA` has the lowest test set errors. We conjecture that `EMA` outperforms `Implicit` because the implicit gradient estimator can have error in practice, as the inner optimization problem cannot be solved optimally, especially when f_θ is parameterized with a general neural network (Blondel et al., 2021; Liao et al., 2018). This issue might be resolved by employing a special structure to f_θ such as linearity or convexity. However, such treatment can prevent the use of arbitrary network architecture as f_θ.



(a) Training method ablation results (b) τ ablation results

Figure 8: Training method ablation results. We train each method five times with different random seeds. The bold lines and shadow region illustrate the averages and standard deviations of the training runs. We smooth the curves using a moving average.

We also note that EMA outperforms BPTO. Because BPTO calculates the gradient by chained computation on the computational graph (at the cost of memory usage), it is difficult to claim that the performance degradation is related to the gradient computation. We conjecture this phenomenon is related to the consistency of $\hat{c}$ estimation. It is noteworthy that BPTO is equivalent to CAVIA with a larger inner gradient step. Hence, FOCA is also superior to CAVIA with the same number of inner optimization steps. The results indicate that EMA trains FOCA effectively while allowing us to employ arbitrary network architecture and lower memory usage.

How does τ affect the performance? We investigate the effect of the delay parameter τ in our training scheme. Our training scheme assumes $\hat{c}$ contains enough information to differentiate a specific system. As it is considered an input, the consistency of $\hat{c}$ over the training step becomes crucial to the stable training of f_θ. In this sense, τ controls the degree of consistency of $\hat{c}$ over the training steps. Higher τ (~ 1.0) introduces relatively drastic changes of $\hat{c}$ during training, while smaller τ changes $\hat{c}$ conservatively. Fig. 8b illustrates the test performance of EMA with $\tau = \{1.0, 0.5, 0.1\}$ against BPTO. It is noteworthy that the forward processes of EMA with $\tau = 1.0$ and BPTO are identical, while the backward processes (i.e., the gradient calculation) are different. By comparing EMA ($\tau = 1.0$) and BPTO, we can observe that using a consistent $\hat{c}$ mechanism is crucial for successful training FOCA. Furthermore, as we decrease τ to 0.1, FOCA exhibits better predictive performance.

6 Conclusion

Motivated by classical modeling-and-identification approaches, we introduced in this work a new contextual meta-learning method: FOCA. We separate the role of different parameters into a model, that captures context-invariant behavioral laws, and context, that specializes in finding instance-specific parameters. FOCA jointly learns model and context parameters by solving online bi-level optimization efficiently. Instead of relying on higher-order derivatives, it employs EMA for faster and more stable training. We empirically demonstrated in both static and dynamic regression tasks on complex dynamical systems that our approach outperforms or is competitive to baselines when evaluated on both in- and out-of-distribution contexts. We also performed ablation studies to examine the efficacy of our proposed training method.

The limitation of our work comes from the assumption regarding the target systems that FOCA aims to model. Currently, FOCA seeks to model the dynamical systems where the system is characterized by a common mathematical model f (parameterized by θ) and a context c. If the target systems do not share consistent behavior, it is hard to be characterized by a specific form of an equation. As a result, the performance of FOCA might degrade. In future studies, we plan to develop a more general algorithm that allows the adaptation over the function classes by allowing it to change θ when the common f is not identifiable.

References

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in Neural Information Processing Systems*

(NeurIPS), 32, 2019.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning (ICML)*, pp. 136–145, 2017.

Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your MAML. In *International Conference on Learning Representations (ICLR)*, 2019.

Luca Bertinetto, Joao F Henriques, Philip HS Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations (ICLR)*, 2019.

Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183*, 2021.

Johannes Brandstetter, Daniel E Worrall, and Max Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations (ICLR)*, 2021.

Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *International Conference on Computer Vision (ICCV)*, pp. 9650–9660, 2021.

Dong Chen, Lingfei Wu, Siliang Tang, Xiao Yun, Bo Long, and Yueting Zhuang. Robust meta-learning with sampling noise and label noise via eigen-reptile. In *International Conference on Machine Learning (ICML)*, pp. 3662–3678, 2022.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Workshop on Syntax, Semantics and Structure in Statistical Translation*, pp. 103–111, 2014.

Bryan C Daniels and Ilya Nemenman. Efficient inference of parsimonious phenomenological models of cellular dynamics using s-systems and alternating regression. *PloS One*, 10(3):e0119821, 2015.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

Tristan Deleu, David Kanaa, Leo Feng, Giancarlo Kerg, Yoshua Bengio, Guillaume Lajoie, and Pierre-Luc Bacon. Continuous-time meta-learning with forward mode differentiation. In *International Conference on Learning Representations (ICLR)*, 2021.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, pp. 1126–1135, 2017.

Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional neural processes. In *International Conference on Machine Learning (ICML)*, pp. 1704–1713, 2018.

Jonathan Gordon, Wessel P Bruinsma, Andrew YK Foong, James Requeima, Yann Dubois, and Richard E Turner. Convolutional conditional neural processes. In *International Conference on Learning Representations (ICLR)*, 2020.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:21271–21284, 2020.

Jayesh K Gupta, Kunal Menda, Zachary Manchester, and Mykel Kochenderfer. Structured mechanical models for robot learning and control. In *Learning for Dynamics and Control*, pp. 328–337, 2020.

Stephen W Hawking. Particle creation by black holes. In *Euclidean Quantum Gravity*, pp. 167–188. World Scientific, 1975.

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.

Mirantha Jayathilaka. Enhancing generalization of first-order meta-learning. 2019.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *International Conference on Learning Representations (ICLR)*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Matthieu Kirchmeyer, Yuan Yin, Jeremie Dona, Nicolas Baskiotis, Alain Rakotomamonjy, and Patrick Gallinari. Generalizing to new physical systems via context-informed dynamics model. In *International Conference on Machine Learning (ICML)*, pp. 11283–11301, 2022.

Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

Kimin Lee, Younggyo Seo, Seunghyun Lee, Honglak Lee, and Jinwoo Shin. Context-aware dynamics model for generalization in model-based reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 5757–5766, 2020a.

Seungjun Lee, Haesang Yang, and Woojae Seong. Identifying physical law of hamiltonian systems via meta-learning. In *International Conference on Learning Representations (ICLR)*, 2020b.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. 2021.

Renjie Liao, Yuwen Xiong, Ethan Fetaya, Lisa Zhang, KiJung Yoon, Xaq Pitkow, Raquel Urtasun, and Richard Zemel. Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning (ICML)*, pp. 3082–3091, 2018.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.

Lennart Ljung. Perspectives on system identification. *Annual Reviews in Control*, 34(1):1–12, 2010.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*, 2017.

Alfred J Lotka. Contribution to the theory of periodic reactions. *The Journal of Physical Chemistry*, 14:271–274, 1910.

Kunal Menda, Jean De Becdelievre, Jayesh Gupta, Ilan Kroo, Mykel Kochenderfer, and Zachary Manchester. Scalable identification of partially observed systems with certainty-equivalent em. In *International Conference on Machine Learning (ICML)*, pp. 6830–6840, 2020.

Zihang Meng, Sathya N Ravi, and Vikas Singh. Physarum powered differentiable linear programming layers and applications. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 8939–8949, 2021.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *International Conference on Learning Representations (ICLR)*, 2018.

Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2018.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, et al. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

Michael Poli, Stefano Massaroli, Federico Berto, Jinkyoo Park, Tri Dao, Christopher Re, and Stefano Ermon. Transform once: Efficient operator learning in frequency domain. In *International Conference on Machine Learning (ICML)*, AI for Science Workshop, 2022.

Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. In *International Conference on Learning Representations (ICLR)*, 2019.

Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-learning with implicit gradients. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International Conference on Machine Learning (ICML)*, pp. 1842–1850, 2016.

Erwin Schrödinger. An undulatory theory of the mechanics of atoms and molecules. *Physical Review*, 28(6):1049, 1926.

George Gabriel Stokes. On the effect of the internal friction of fluids on the motion of pendulums. *Transaction of the Cambridge Philosophical Society*, 9:8–106, 1851.

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 (NeurIPS)*, 30, 2017.

Yuan Yin, Ibrahim Ayed, Emmanuel de Bézenac, Nicolas Baskiotis, and Patrick Gallinari. Leads: Learning dynamical systems that generalize across environments. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:7561–7573, 2021.

Pan Zhou, Xiaotong Yuan, Huan Xu, Shuicheng Yan, and Jiashi Feng. Efficient meta learning via minibatch proximal update. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1): 43–76, 2020.

Luisa Zintgraf, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning (ICML)*, pp. 7693–7702, 2019.

A DETAILS OF STATIC REGRESSION EXPERIMENTS

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

Model architecture We employ `Encoder`, `CAVIA`, and `CoDA` as the baselines. 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 `MLP`(n_1, n_2, ..., n_l; `act`). We refer to a cross multi-head attention block (Vaswani et al., 2017) with h heads and x hidden dimensions as `X.MHA`($h \times x$).

Table A.1: Polynomial regression model architectures

	Context encoder / parameter generator	f_θ	Inner step K	Inner step size α	τ
`Encoder`	`MLP`(2,32)/`MLP`(1,32)-`X.MHA`(4×32)	`MLP`(1+32, 64, 32, 1;`LeakyReLU`)	–	–	–
`CAVIA`	–	`MLP`(1+32, 64, 32, 1;`LeakyReLU`)	5	1	–
`CoDA`	$\boldsymbol{W} : \mathbb{R}^{\lvert c\rvert} \to \mathbb{R}^{\lvert\theta\rvert}$	`MLP`(1, 64, 32, 1;`LeakyReLU`)	500	0.001	–
`FOCA`	–	`MLP`(1+32, 64, 32, 1;`LeakyReLU`)	100	0.001	0.1

Table A.1 summarizes the network architectures. The green colored values indicate the dimension of context $\hat{c}$. For `CAVIA`, we also perform hyperparameter search to optimize K. We found that `CAVIA` with $K > 5$ underperforms, as compared to $K = 5$. For `CoDA`, we set the dimension of context as 2 (i.e., $|c| = 2$), following the default setting of Kirchmeyer et al. (2022).

Training details We train all models with mini-batches of 256 polynomials for 4,048 epochs using Adam (Kingma & Ba, 2015) with an initial learning rate of 0.001. The learning rate is scheduled by the cosine annealing method (Loshchilov & Hutter, 2017).

B DETAILS OF MASS SPRING (MS) EXPERIMENTS

In this section, we provide the model architectures, training process, and additional experimental results for the mass-spring systems.

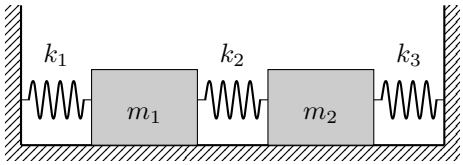


Figure A.1: Target mass-spring system. The models require to adapt to the change of spring constants (k_1, k_2, k_3) and masses (m_1, m_2).

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 second-order ODE:

$$\begin{aligned} \frac{\mathrm{d}^2 x_1}{\mathrm{d}t^2} &= -\frac{k_1 + k_2}{m_1} x_1 + \frac{k_2}{m_1} x_2 \\ \frac{\mathrm{d}^2 x_2}{\mathrm{d}t^2} &= -\frac{k_2 + k_3}{m_2} x_2 + \frac{k_2}{m_2} x_1 \end{aligned} \tag{A.1}$$

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

Data generation For training data generation, we generate 128 mass-spring systems whose parameters $(k_1, k_2, k_3, m_1, m_2)$ are sampled from $\mathcal{U}(0.75, 1.25)^5$ and numerically solve the mass-spring systems with Runge–Kutta 45 for $T = 10$ seconds with $\Delta t = 0.15$ second time intervals.

Model architecture For f_θ, we employ the 1D-CNN model from Brandstetter et al. (2021), which stacks a MLP, 1D-CNN, and consistency decoder (Brandstetter et al., 2021) with the bundling parameter $N = 25$. 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 (Cho et al., 2014) with hidden dimension x as `GRU`(x).

Table A.2: Mass-spring prediction model architectures

	Context encoder / parameter generator	f_θ
`Encoder`	`GRU(64)-MLP(64, 64)`	`MLP(4×25+64, 64, 32, 4×25;LeakyReLU)-Conv(4, 4, 8)-LeakyReLU-Conv(4,4,1)-C.Dec`
`CAVIA`	–	`MLP(4×25+64, 64, 32, 4×25;LeakyReLU)-Conv(4, 4, 8)-LeakyReLU-Conv(4,4,1)-C.Dec`
`CoDA`	$\boldsymbol{W} : \mathbb{R}^{\lvert c\rvert} \to \mathbb{R}^{\lvert\theta\rvert}$	`MLP(4×25, 64, 32, 4×25;LeakyReLU)-Conv(4, 4, 8)-LeakyReLU-Conv(4,4,1)-C.Dec`
`FOCA`	–	`MLP(4×25+64, 64, 32, 4×25;LeakyReLU)-Conv(4, 4, 8)-LeakyReLU-Conv(4,4,1)-C.Dec`

Table A.2 summarizes the network architectures. The green colored values indicate the dimension of context $\hat{c}$. We set the inner step K and step size α as 100/1 and 0.001/1 for `FOCA` and `CAVIA`, respectively, and τ as 0.1. For `CoDA`, we set the dimension of context as 2 (i.e., $|c| = 2$), following the default setting of Kirchmeyer et al. (2022).

Training details We train all models with mini-batches of size 512 for 1,000 epochs using Adam (Kingma & Ba, 2015) with the initial learning rate of 0.001 and pushforward regularization (Brandstetter et al., 2021). We use the past observations from the previous $2N$ to N steps as the input of the adaptation process. For CoDA, we performed a hyperparemter search and found that $L1$ regularization on the norm of c as 1e-3 and $\boldsymbol{W}$ as 1e-7 performs well.

Evaluation setting For in-training evaluations, we generate 2,048 mass-spring systems whose parameters $(k_1, k_2, k_3, m_1, m_2)$ are sampled from $\mathcal{U}(0.75, 1.25)^5$ and numerically solve mass-spring system with Runge–Kutta 45 with $T = 5.0$ and $\Delta t = 0.01$. For out-of-training evaluations, we generate 2,048 mass-spring systems whose parameters $(k_1, k_2, k_3, m_1, m_2)$ are sampled from $\mathcal{U}(0.60, 0.75)^5 \cup \mathcal{U}(1.25, 1.35)^5$ for $T = 5.0$ with $\Delta t = 0.01$. All models take the first and second 0.25 seconds of observation for the task adaptation and predict 4.5 seconds future states via the model rollout.

Additional results We provide the visualization of the model generalization errors to the change of m_1, m_2 on the mass-spring systems, as shown in Fig. A.2.

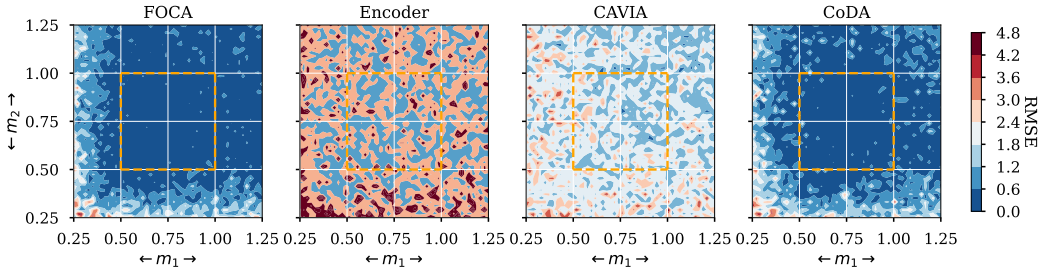


Figure A.2: In/out-of-distribution losses (in RMSE) on MS. Two parameters of MS (m_1, m_2) are generated from $[0.25, 1.25]^2$ while the other two parameters are fixed to 0.75. The orange box indicates the boundaries of the training parameter distribution.

C Details of Lotka-Volterra (LV) experiments

In this section, we provide the model architectures, training process, and additional experimental results for the Lokta-Volterra systems (Lotka, 1910).

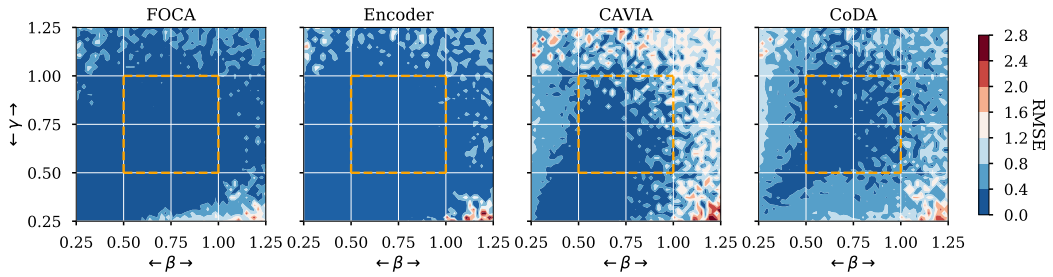


Figure A.3: In/out-of-distribution losses (in RMSE) on LV. Two parameters of LV (β, γ) are generated from $[0.25, 1.25]^2$ while the other two parameters are fixed to 0.5. The orange box indicates the boundaries of the training parameter distribution.

The Lokta-Volterra system describes the interaction between a prey-predator pair in an ecosystem, formalized into the following ODE:

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= \alpha x - \beta xy, \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= \delta xy - \gamma y, \end{aligned} \tag{A.2}$$

where x, y are respectively the quantity of the prey and the predator and $\alpha, \beta, \gamma, \delta$ define how two species interact.

Data generation For training data generation, we generate 128 Lokta-Volterra systems whose parameters $(\alpha, \delta) = (0.5, 0.5)$ and $(\beta, \gamma) \sim \mathcal{U}(0.5, 1.0)^2$ following Kirchmeyer et al. (2022). We then numerically solve the systems with Runge–Kutta 45 for $T = 50.0$ seconds with $\Delta t = 0.5$ second time intervals.

Model architecture We employ the same network architectures of the MS experiments except with a different input dimensions of 2 and bundling parameter $N = 20$.

Training details We train all models with mini-batches of size 128 for 5,000 epochs. Other details are kept the same as mass-spring experiments.

Evaluation setting For in-training evaluations, we generate 2,048 Lokta-Volterra systems similarly to the training data generation. For out-of-training evaluations, we generate 2,048 Lokta-Volterra systems whose parameters $(\alpha, \delta) = (0.5, 0.5)$ and $(\beta, \gamma) \sim \mathcal{U}(0.25, 0.5)^2 \cup \mathcal{U}(1.0, 1.25)^2$ for $T = 50.0$ with $\Delta t = 0.5$. All models take the first and second 10.0 seconds of observation for the task adaptation and predict 30.0 seconds future states via the model rollout.

D Details of Glycolytic Oscillator (GO) Experiments

In this section, we provide the model architectures, training process, and additional experimental results for the glycolytic oscillators (Daniels & Nemenman, 2015).

The glycolytic oscillators describe yeast glycolysis dynamics with the following ODE:

$$
\begin{aligned}
\frac{\mathrm{d}S_1}{\mathrm{d}t} &= J_0 - \frac{k_1 S_1 S_6}{1 + (1/K_1^q) S_6^q} \\
\frac{\mathrm{d}S_2}{\mathrm{d}t} &= 2\frac{k_1 S_1 S_6}{1 + (1/K_1^q) S_6^q} - k_2 S_2 (N - S_5) - k_6 S_2 S_5 \\
\frac{\mathrm{d}S_3}{\mathrm{d}t} &= k_2 S_2 (N - S_5) - k_3 S_3 (A - S_6) \\
\frac{\mathrm{d}S_4}{\mathrm{d}t} &= k_3 S_3 (A - S_6) - k_4 S_4 S_5 - \kappa (S_4 - S_7) \\
\frac{\mathrm{d}S_5}{\mathrm{d}t} &= k_2 S_2 (N - S_5) - k_4 S_4 S_5 - k_6 S_2 S_5 \\
\frac{\mathrm{d}S_6}{\mathrm{d}t} &= -2\frac{k_1 S_1 S_6}{1 + (1/K_1^q) S_6^q} + 2k_3 S_3 (A - S_6) - k_5 S_6 \\
\frac{\mathrm{d}S_7}{\mathrm{d}t} &= \psi \kappa (S_4 - S_7) - k S_7,
\end{aligned}
\tag{A.3}
$$

where $S_1, S_2, S_3, S_4, S_5, S_6, S_7$ (states) represent the concentrations of 7 biochemical species and $J_0, k_1, k_2, k_3, k_4, k_5, k_6, K_1, q, N, A, \kappa, \psi$ and k are the parameters of the glycolytic oscillators.

Data generation For training data generation, we generate 128 glycolytic oscillators with the fixed parameters $J_0 = 2.5, k_2 = 6, k_3 = 16, k_4 = 100, k_5 = 1.28, k_6 = 12, q = 4, N = 1, A = 4, \kappa = 13, \psi = 0.1$ and $k = 1.8$ by sampling integer $k_1 \sim \mathcal{U}(80, 100)$ and $K_1 \sim \mathcal{U}(0.5, 1.0)$. We adopt the values or ranges of the parameters from Kirchmeyer et al. (2022). We then numerically solve the systems with Runge–Kutta 45 for $T = 5.0$ seconds with $\Delta t = 0.05$ second time intervals.

Model architecture We employ the same network architectures of the MS experiments except with a different input dimensions of 7 and bundling parameter $N = 10$. For `CoDA`, we set the context dimension as 3.

Training details We train all models with mini-batches of size 512 for 5,000 epochs. Other details are kept the same as mass-spring experiments.

Evaluation setting For in-training evaluations, we generate 2,048 glycolytic oscillators similarly to the training data generation. For out-of-training evaluations, we generate 2,048 glycolytic oscillators whose parameters $k_1 \sim \mathcal{U}(75, 80) \cup \mathcal{U}(100, 105)$ and $K_1 \sim \mathcal{U}(0.45, 0.5) \cup \mathcal{U}(1.00, 1.05)$ and $(J_0, k_2, k_3, k_4, k_5, k_5, k_6, q, N, A, \kappa, \psi, k) = (2.5, 6, 16, 100, 1.28, 12, 4, 1, 4, 13, 0.1, 1.8)$ for $T = 5.0$ with $\Delta t = 0.05$. All models take the first and second 0.5 seconds of observation for the task adaptation and predict 4.5 seconds future states via the model rollout.

Additional results We provide the visualization of the model generalization errors to the change of k_1, K_1 on the glycolytic oscillators, as shown in Fig. A.4.

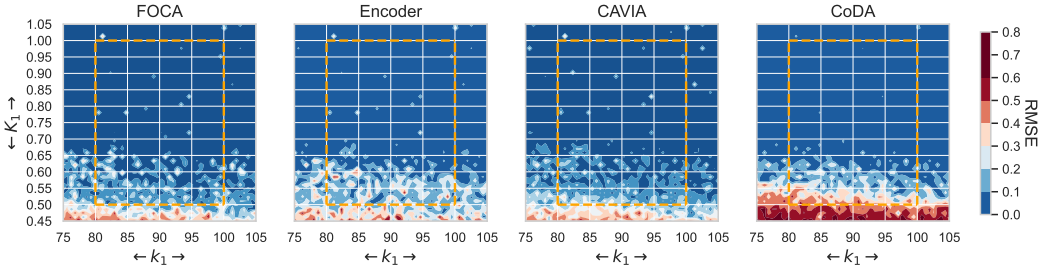


Figure A.4: In/out-of-distribution losses (in RMSE) on GO. The orange box indicates the boundaries of the training parameter distribution.

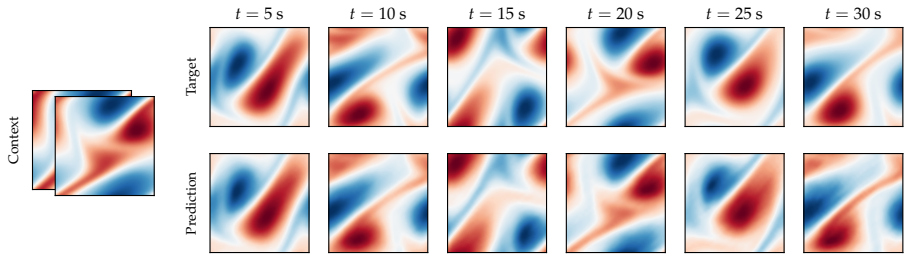


Figure A.5: Out-of-distribution generalization on the Navier-Stokes equations: FOCA is trained with viscosity $\nu \in [8 \times 10^{-4}, 1.2 \times 10^{-3}]$ and tested with $\nu = 6 \times 10^{-4}$. The model can reconstruct states even after long autoregressive rollouts in unseen conditions.

E Details of Navier-Stokes (NS) experiments

In this section, we provide the model architectures, training process, and additional experimental results for modeling the 2D Navier-Stokes equations (Stokes, 1851).

The Navier-Stokes equations describe the dynamics of incompressible flows with a two-dimensional PDE. In vorticity form they can be written as:

$$\begin{aligned} \frac{\partial w}{\partial t} &= -v \nabla w + \nu \Delta w + f \\ \nabla v &= 0 \\ w &= \nabla \times v \end{aligned} \tag{A.4}$$

where v is the velocity field and w is the vorticity, ν is the viscosity, and f is a forcing term. The domain is subject to periodic boundary conditions.

Data generation We generate trajectories with a temporal resolution of $\Delta t = 1$ and a time horizon of $t = 10$. The space is discretized on a 32×32 grid and we set $f(x, y) = 0.1(\sin(2\pi(x + y)) + \cos(2\pi(x + y)))$, where x, y are coordinates on the discretized domain (Yin et al., 2021). For training data, similarly to Kirchmeyer et al. (2022), we consider 5 training environments with $\nu \in \{8 \cdot 10^{-4}, 9 \cdot 10^{-4}, 1.0 \cdot 10^{-3}, 1.1 \cdot 10^{-3}, 1.2 \cdot 10^{-3}\}$ respectively. Each environment contains a total of 100 different initial conditions for a total of 500 training sequences.

Model architecture For this experiment, we employ the Fourier Neural Operator (FNO) (Li et al., 2021). Thanks to their layers in the spectral domain and frequency mode pruning, frequency domain models have been proven to be well suited to model complex dynamical systems characterized by a multitude of natural frequencies (Pathak et al., 2022; Poli et al., 2022). We employ an FNO model with 4 spectral convolution layers, hidden layers with width 10, and frequency mode pruning set to the 12 highest frequencies. Temporal bundling is set to $N = 1$. We set the context dimension of `CoDA` to 3. For models whose input includes the context, i.e. `Encoder`, `CAVIA` and `FOCA`, we keep the same context dimension $|\hat{c}| = 64$ as in the previous experiments; the inferred context passes through a linear layer and resized to the grid size of 32×32 as an additional input channel of the FNO.

Training details We train all models with mini-batches of size 16 for 100 epochs. Other details are kept the same as in the mass-spring experiments.

Evaluation setting For in-training adaptation, we consider 4 environments with viscosity $\nu \in \{8.5 \cdot 10^{-4}, 9.5 \cdot 10^{-4}, 1.05 \cdot 10^{-3}, 1.15 \cdot 10^{-3}\}$ while for out-of-distribution adaptation we generate data from 4 environments with unseen ranges of viscosity, i.e. $\nu \in \{7.0 \cdot 10^{-4}, 7.5 \cdot 10^{-4}, 1.25 \cdot 10^{-3}, 1.30 \cdot 10^{-3}\}$. Each adaptation environment is generated out of 30 different initial conditions. All the models take as input the first two steps (i.e. the first 2 seconds) and predict the rest of the sequence with autoregressive rollouts.

Additional results We provide in Fig. A.5 an additional qualitative visualization showing the out-of-distribution generalization on the Navier-Stokes equations. FOCA is trained with viscosity $\nu \in [8 \times 10^{-4}, 1.2 \times 10^{-3}]$ and tested with $\nu = 6 \times 10^{-4}$, demonstrating its adaptation capability in challenging and unseen settings.