Fast Unsupervised Deep Outlier Model Selection with Hypernetworks

Xueying Ding
xding2@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Yue Zhao
yzhao010@usc.edu
University of Southern California
Los Angeles, CA, USA

Leman Akoglu
lakoglu@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Abstract

Deep neural network based Outlier Detection (DOD) has seen a recent surge of attention thanks to the many advances in deep learning. In this paper, we consider a critical-yet-understudied challenge with unsupervised DOD, that is, effective hyperparameter (HP) tuning/model selection. While several prior work report the sensitivity of OD models to HP settings, the issue is ever so critical for the modern DOD models that exhibit a long list of HPs. We introduce HyPer for tuning DOD models, tackling two fundamental challenges: (1) validation without supervision (due to lack of labeled outliers), and (2) efficient search of the HP/model space (due to exponential growth in the number of HPs). A key idea is to design and train a novel hypernetwork (HN) that maps HPs onto optimal weights of the main DOD model. In turn, HyPer capitalizes on a *single* HN that can dynamically generate weights for *many* DOD models (corresponding to varying HPs), which offers significant speed-up. In addition, it employs meta-learning on historical OD tasks with labels to train a proxy validation function, likewise trained with our proposed HN efficiently. Extensive experiments on different OD tasks show that HyPer achieves competitive performance against 8 baselines with significant efficiency gains.

CCS Concepts

• **Computing methodologies → Machine learning algorithms**;

Keywords

Deep Outlier Detection, Unsupervised Model Selection, Hyperparameter Tuning, Hypernetworks, Meta-learning

ACM Reference Format:
Xueying Ding, Yue Zhao, and Leman Akoglu. 2024. Fast Unsupervised Deep Outlier Model Selection with Hypernetworks. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3637528.3672003

1 Introduction

Motivation. With advances in deep learning, deep neural network (NN) based outlier detection (DOD) has seen a surge of attention in recent years [28, 31]. These models, however, inherit many hyperparameters (HPs) that can be organized three ways; architectural

KDD '24, August 25–29, 2024, Barcelona, Spain
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3672003

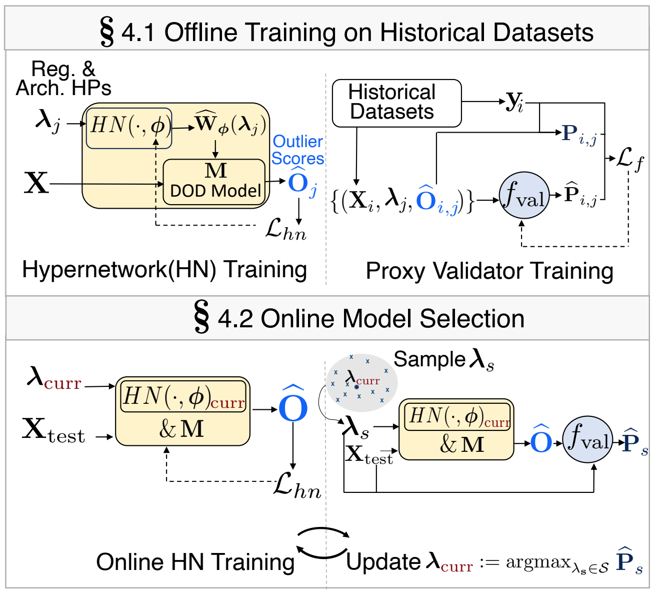

§ 4.1 Offline Training on Historical Datasets
Reg. & Arch. HPs
$\boldsymbol{\lambda}_j$
$HN(\cdot, \phi)$
$\widehat{\mathbf{W}}_\phi(\boldsymbol{\lambda}_j)$
$\mathbf{X}$
M
DOD Model
Outlier Scores
$\widehat{\mathbf{O}}_j$
$\mathcal{L}_{hn}$
Hypernetwork(HN) Training
Historical Datasets
$\mathbf{y}_i$
$\mathbf{P}_{i,j}$
$\mathcal{L}_f$
$\{(\mathbf{X}_i, \boldsymbol{\lambda}_j, \widehat{\mathbf{O}}_{i,j})\}$
f_{val}
$\widehat{\mathbf{P}}_{i,j}$
Proxy Validator Training
§ 4.2 Online Model Selection
$\boldsymbol{\lambda}_{\text{curr}}$
$\mathbf{X}_{\text{test}}$
$HN(\cdot, \phi)_{\text{curr}}$
$\& \mathbf{M}$
$\widehat{\mathbf{O}}$
$\mathcal{L}_{hn}$
Sample $\boldsymbol{\lambda}_s$
$\boldsymbol{\lambda}_{\text{curr}}$
$\boldsymbol{\lambda}_s$
$\mathbf{X}_{\text{test}}$
$HN(\cdot, \phi)_{\text{curr}}$
$\& \mathbf{M}$
$\widehat{\mathbf{O}}$
f_{val}
$\widehat{\mathbf{P}}_s$
Online HN Training
Update $\boldsymbol{\lambda}_{\text{curr}} := \text{argmax}_{\lambda_s \in \mathcal{S}} \widehat{\mathbf{P}}_s$

Figure 1: HyPer framework illustrated. (top) Offline meta-training of f_{val} (depicted in) on historical datasets for proxy validation (§4.1); (bottom) Online model selection on a new dataset (§4.2). We accelerate both meta-training and model selection using hypernetworks (HN) (depicted in ; §3.1).

(e.g. depth, width), regularization (e.g. dropout rate, weight decay), and optimization HPs (e.g. learning rate). As expected, their performance is highly sensitive to the HP settings [6]. This makes effective HP or model selection critical, yet computationally costly as the model space grows exponentially large in the number of HPs.

Hyperparameter optimization (HPO) can be written as a bilevel problem, where the optimal parameters $\mathbf{W}^*$ (i.e. NN weights) on the training set depend on the hyperparameters $\boldsymbol{\lambda}$.

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda}} \; \mathcal{L}_{\text{val}}(\boldsymbol{\lambda}; \mathbf{W}^*) \quad s.t. \quad \mathbf{W}^* = \arg\min_{\mathbf{W}} \; \mathcal{L}_{\text{trn}}(\mathbf{W}; \boldsymbol{\lambda}) \qquad (1)$$

where $\mathcal{L}_{\text{val}}$ and $\mathcal{L}_{\text{trn}}$ denote the validation and training losses, respectively. There is a body of literature on HPO for supervised settings [2, 16, 35], as well as for OD that use labeled outliers for validation [15, 17, 18]. While supervised model selection leverages $\mathcal{L}_{\text{val}}$, unsupervised OD posits a unique challenge: it does not exhibit labeled hold-out data to evaluate $\mathcal{L}_{\text{val}}$. It is unreliable to employ the same loss $\mathcal{L}_{\text{trn}}$ as $\mathcal{L}_{\text{val}}$ as models with minimum training loss do not necessarily associate with accurate detection [6].

Prior work. Earlier work have proposed intrinsic measures for unsupervised model evaluation, based on input data and output (outlier scores) characteristics [9, 24], using internal consensus among various models [7, 19], as well as properties of the learned weights

[25]. As recent meta-analyses have shown, such intrinsic measures are quite noisy; only slightly and often no better than random [20]. Moreover, they suffer from the exponential compute cost in large HP spaces as they require training numerous candidate models for evaluation. More recent solutions leverage meta-learning by selecting a model for a new dataset based on similar historical datasets [43, 44] . They are likewise challenged computationally for large HP spaces and cannot handle any continuous HPs. Our proposed HyPer also leverages meta-learning, while it is more efficient with the help of hypernetworks as well as more effective by handling continuous HPs with a better-designed proxy validation function.

Present Work. We introduce HyPer for unsupervised and efficient hyperparameter tuning for deep-NN based OD. HyPer tackles both of two key challenges with unsupervised DOD model selection: **(Ch1)** lack of supervision, and **(Ch2)** scalability as tempered by the cost of training numerous candidate models. For **Ch1**, we employ a meta-learning approach, where we train a proxy validation function, f_{val}, that maps the HPs $\boldsymbol{\lambda}$, input data, and output outlier scores of DOD models onto corresponding detection performance on historical tasks, as illustrated in Fig. 1 (top, right). Since meta-learning builds on past experience (historical datasets) *with* labels, the performance of various models can be evaluated.

Having substituted $\mathcal{L}_{\text{val}}$ with meta-trained f_{val}, one can adopt existing supervised HPO solutions [8] toward model selection for a given/new dataset *without* labels. However, most of those are susceptible to the scalability challenge (**Ch2**), as they train each candidate model (with varying $\boldsymbol{\lambda}$) independently from scratch. To address scalability, and bypass the expensive process of fully training each candidate separately, we leverage *hypernetworks* (HN). This idea is inspired by the self-tuning networks (STN) [22], which estimate the "best-response" function that maps HPs onto optimal weights through a parameterized hypernetwork (HN), i.e. $\widehat{\mathbf{W}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda}) \approx \mathbf{W}^*$. A single auxiliary HN model can generate the weights of the main DOD model with varying HPs. In essence, it learns how the model weights should change or *respond to* the changes in the HPs (hence the name, best-response), illustrated in Fig. 1 (top, left). As a key contribution, we go beyond just the regularization HPs (e.g., dropout rate) that STN [22] considered, but propose a novel HN model that can also respond to the *architectural* HPs; including depth and width for DOD models with fully-connected layers.

When a new test dataset (*without* labels) arrives, HyPer jointly optimizes the HPs $\boldsymbol{\lambda}$ and the HN parameters $\boldsymbol{\phi}$ in an alternating fashion, as shown in Fig. 1 (bottom). Over iterations, it alternates between (1) **HN training** that updates $\boldsymbol{\phi}$ to approximate the best-response in a local neighborhood around the current hyperparameters via $\mathcal{L}_{\text{trn}}$, and then (2) **HP optimization** that updates $\boldsymbol{\lambda}$ in a gradient-free fashion by estimating detection performance through f_{val} of a large set of candidate $\boldsymbol{\lambda}$'s sampled from the same neighborhood, using the corresponding HN-generated weights efficiently.

Contributions. HyPer addresses the model selection problem for *unsupervised* deep-NN based outlier detection (DOD), applicable to any DOD model, and is *efficient* in the face of the large continuous HP space, tackling both **Ch1** and **Ch2**. HyPer's notable efficiency is thanks to our proposed hypernetwork (HN) model that generates DOD model parameters (i.e. NN weights) in response to changes in the HPs associated with regularization as well as NN architecture—in effect, employing a single HN that can act like many DOD models. Further, it offers unsupervised tuning thanks to a proxy validation function trained via meta-learning on historical tasks, which also benefits from the efficiency of our HN.

We compare HyPer against 8 baselines ranging from simple to state-of-the-art (SOTA) through extensive experiments on 35 benchmark datasets using autoencoder based DOD. HyPer offers the best performance-runtime trade-off, leading to statistically better detection than most baselines; e.g., 2× ↑ over default HP in PyOD [42]), and 4× speed-up against the SOTA approach ELECT [44].

Accessibility and Reproducibility. We open-source all code and datasets at https://github.com/xyvivian/HYPER.

2 Problem and Preliminaries

The sensitivity of outlier detectors to the choice of their hyperparameters (HPs) is well documented [1, 4, 20]. Deep-NN based OD models are no exception, if not even more vulnerable to HP configuration [6], as they exhibit a long list of HPs; architectural, regularization and optimization HPs. In fact, it would not be an overstatement to point to unsupervised outlier model selection as the primary obstacle to unlocking the ground-breaking potential of deep-NNs for OD. This is exactly the problem we consider.

Problem 1 (Unsupervised Deep Outlier Model Selection (UDOMS)). Given *a new input dataset (i.e., detection task)* $\mathcal{D}_{test} = (\mathbf{X}_{test}, \emptyset)$ *without any labels, and a deep-NN based OD model M;* Output *model parameters corresponding to a selected hyperparameter/model configuration* $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ *(where* $\boldsymbol{\Lambda}$ *is the model space) to employ on* $\mathbf{X}_{test}$ *to maximize M's detection performance.*

Key Challenges: Our work addresses two key challenges that arise when tuning deep neural network models for outlier detection: (**Ch1**) *Validation without supervision* & (**Ch2**) *Large HP/model space.*

First, unsupervised OD does not exhibit any labels and therefore model selection via validating detection performance on labeled hold-out data is not possible. While model parameters can be estimated end-to-end through unsupervised training losses, such as reconstruction error or one-class losses, one cannot reliably use the same loss as the validation loss; in fact, low error could easily associate with poor detection since most DOD models use point-wise errors as their outlier scores.

Second, model tuning for the modern OD techniques based on deep-NNs with many HPs is a much larger scale ball-game than that for their shallow counterparts with only 1-2 HPs. This is both due to their (*i*) large number of HPs and also (*ii*) longer training time they typically demand. In other words, the model space that is exponential in the number of HPs and the costly training of individual models necessitate efficient strategies for search.

3 Fast and Unsupervised: Key Building Blocks of HyPer

To address the above challenges in UDOMS, we propose two primary building blocks for HyPer: (1) hypernetworks for fast model weight prediction (§3.1) and (2) proxy validator f_{val} that transfers supervision to evaluate model performance on a new dataset without labels (§3.2). In §4, we describe how to put together these building blocks toward fast and unsupervised deep OD model selection.

3.1 Hypernetwork: Train One, Get Many

To tackle the challenge of model-building efficiency, we propose a version of hypernetworks (HN) that can efficiently train DOD models with different hyperparameter configurations. A hypernetwork (HN) is a network generating weights (i.e. parameters) for another network (in our case, the DOD model) [11]. Our input to HN is $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$, which is one HP configuration and breaks down into two components as $\boldsymbol{\lambda} = [\boldsymbol{\lambda}_{reg}, \boldsymbol{\lambda}_{arch}]$, corresponding to regularization HPs (e.g. dropout, weight decay) and architectural HPs (number of layers and width of each layer). Parameterized by ϕ, the HN maps a specific hyperparameter configuration $\boldsymbol{\lambda}_j$ to DOD model weights $\widehat{\mathbf{W}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda}_j) := HN(\boldsymbol{\lambda}_j; \phi)$, which parameterize the DOD model for hyperparameter configuration $\boldsymbol{\lambda}_j$.

We propose changes to the HN [11], such that (1) the output $\widehat{\mathbf{W}}_{\boldsymbol{\phi}}$ can adjust to different architectural shapes, (2) HN can output sufficiently diverse weights in response to varying $\boldsymbol{\lambda}$ inputs, and (3) HN training is more efficient than training individual DOD models.

Architecture Masking. To allow the HN output to adapt to various architectures, we let the output $\widehat{\mathbf{W}}_{\boldsymbol{\phi}}$'s size be equal to size of the largest architecture in model space $\boldsymbol{\Lambda}$. Then for each $\boldsymbol{\lambda}_{arch}$, we build a corresponding architecture masking and feed the masked-version of $\widehat{\mathbf{W}}_{\boldsymbol{\phi}}$ to the DOD model. In other words, the output $\widehat{\mathbf{W}}_{\boldsymbol{\phi}}$ handles all smaller architectures by properly padding zeros.

Taking DOD models built upon MLPs as an example (see Fig. 2), we make HN output $\widehat{\mathbf{W}}_{\boldsymbol{\phi}} \in \mathbb{R}^{(D \times W \times W)}$, where D and W denote the maximum depth and maximum layer width from $\boldsymbol{\Lambda}$. Assume $\boldsymbol{\lambda}_{arch}$ contains the abstraction of a smaller architecture; e.g., L layers with corresponding width values $\{W_1, W_2 \ldots, W_L\}$ all less than or equal to W. The architectural HP $\boldsymbol{\lambda}_{arch} \in \mathbb{N}^D$ is defined as:

$$\lambda_{\text{arch}} = [W_1, W_2, \ldots, W_{\lfloor L/2 \rfloor}, \underbrace{0, \ldots, 0}_{(D-L)\text{ zeros}}, W_{\lfloor L/2 \rfloor + 1}, \ldots, W_{(L-1)}, W_L] \,.$$

Then, we construct the architecture masking $\mathbf{A} \in \{0,1\}^{(D \times W \times W)}$:

$$\begin{cases} \mathbf{A}_{[l, 0:\boldsymbol{\lambda}_{arch}[0], :]} = 1 & \text{, if } l = 0 \\ \mathbf{A}_{[l, 0:\boldsymbol{\lambda}_{arch}[l], 0:\boldsymbol{\lambda}_{arch}[l-z]]} = 1 & \text{, otherwise} \end{cases} \tag{2}$$

where $\boldsymbol{\lambda}_{arch}[l-z]$ is the last non-zero entry in $\boldsymbol{\lambda}_{arch}[0:l]$. (e.g., for $\boldsymbol{\lambda}_{arch} = [5,3,0,0,3]$ and $l = 4$, the last nonzero entry is $\boldsymbol{\lambda}_{arch}[1]$ where $z = 3$.) Then, l'th layer weights are multiplied by masking as $\mathbf{A}_{[l,:,:]} \odot \widehat{\mathbf{W}}_{\boldsymbol{\phi},l}$, where non-zero entries are of shrunk dimensions. If $\mathbf{A}_{[l,:,:]}$ contains only zeros, layer weights become all zeros, representing a "No operation" and is ignored in the DOD model.

We find that this masking works well with linear autoencoders with a "hourglass" structure, in which case the maximum width W is the input dimension. For convolutional networks, even though we can tune depths and channels, we can also include kernel sizes and dilation rate by properly padding $\widehat{\mathbf{W}}_{\boldsymbol{\phi}}$ with zeros [38]. We make HN output $\widehat{\mathbf{W}}_{\boldsymbol{\phi}} \in \mathbb{R}^{(D \times M_{ch} \times M_{ch} \times M_k \times M_k)}$, where D, M_{ch}, M_k represent maximum number of layers, channels, and kernel size specified in $\boldsymbol{\Lambda}$, respectively.

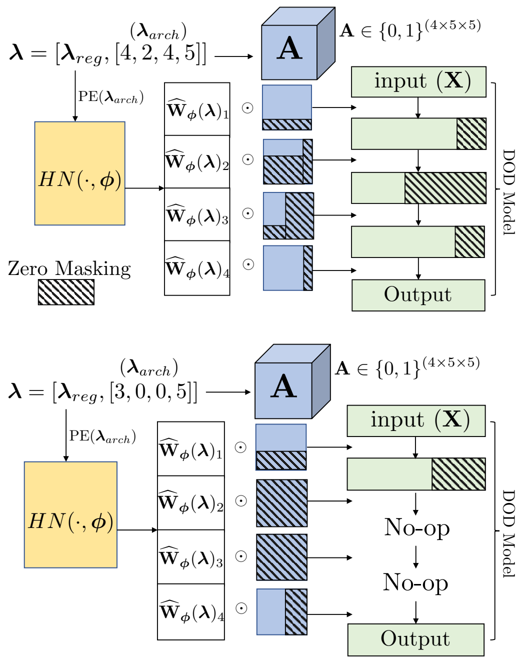
Assume $\boldsymbol{\lambda}_{arch}$ contains the abstraction of a smaller architectures, e.g. L layers with corresponding channel values $\{M_{c1}, M_{c2}, ..., M_{cL}\}$ all less than or equal to M_{ch}, and $\{K_1, K_2, ..., K_L\}$ are less than or equal to M_k. Then, the $\boldsymbol{\lambda}_{arch} \in \mathbb{N}^{2D}$ is given as:

$$\boldsymbol{\lambda}_{arch} = [M_{c1}, K_1, M_{c2}, K_2, \ldots, M_{c\lfloor L/2 \rfloor}, K_{\lfloor L/2 \rfloor}, \underbrace{0, \ldots, 0}_{2(D-L)\text{ zeros}}, M_{c\lfloor L/2 \rfloor + 1}, K_{\lfloor L/2 \rfloor + 1}, \ldots, M_{cL}, K_L] \tag{3}$$

The architecture masking $\mathbf{A} \in \{0,1\}^{(D \times M_{ch} \times M_{ch} \times M_k \times M_k)}$ is constructed as the following:

$$\begin{cases} \mathbf{A}_{[l, 0:\boldsymbol{\lambda}_{arch}[2\times l], :, \lfloor \frac{M_{ch}}{2} \rfloor - \lfloor \frac{\boldsymbol{\lambda}_{arch}[2\times l+1]}{2} \rfloor : \lfloor \frac{M_{ch}}{2} \rfloor + \lfloor \frac{\boldsymbol{\lambda}_{arch}[2\times l+1]}{2} \rfloor]} = 1, \text{ if } l = 0 \\ \mathbf{A}_{[l, 0:\boldsymbol{\lambda}_{arch}[2\times l], 0:\boldsymbol{\lambda}_{arch}[2\times(l-z)], \lfloor \frac{M_{ch}}{2} \rfloor - \lfloor \frac{\boldsymbol{\lambda}_{arch}[2\times l+1]}{2} \rfloor : \lfloor \frac{M_{ch}}{2} \rfloor + \lfloor \frac{\boldsymbol{\lambda}_{arch}[2\times l+1]}{2} \rfloor]} \\ \qquad = 1, \text{ otherwise} \end{cases} \tag{4}$$

Again, $\boldsymbol{\lambda}_{arch}[2\times(l-z)]$ is the last entry corresponding to the non-zero input channel in $\boldsymbol{\lambda}_{arch}[2\times l]$. Similar to the linear operation, at layer l, if $\boldsymbol{\lambda}_{arch}[2 \times l]$ is all zero, then the resulting $\mathbf{A}_{[l,:,:,:,:]}$ would contain only zeros and represent a "No-op" in the DOD model. Otherwise, assume we want obtain a smaller kernel size, $K_l \le M_k$ at layer l, the corresponding $\mathbf{A}_{[l,:,:,:,:]}$ pads zeros around the size $M_{ch} \times k \times k$ center. The masked weights $\mathbf{A}_{[l,:,:,:,:]} \odot \widehat{\mathbf{W}}_{\boldsymbol{\phi},l}$ are equivalent to obtaining smaller-size kernel weights. Notice that, when kernel sizes are different, the output of the layer's operation will also



$(\boldsymbol{\lambda}_{arch})$
$\boldsymbol{\lambda} = [\boldsymbol{\lambda}_{reg}, [4,2,4,5]]$
A
$\mathbf{A} \in \{0,1\}^{(4\times5\times5)}$
PE($\boldsymbol{\lambda}_{arch}$)
$HN(\cdot,\phi)$
$\widehat{\mathbf{W}}_{\phi}(\boldsymbol{\lambda})_1$
$\widehat{\mathbf{W}}_{\phi}(\boldsymbol{\lambda})_2$
$\widehat{\mathbf{W}}_{\phi}(\boldsymbol{\lambda})_3$
$\widehat{\mathbf{W}}_{\phi}(\boldsymbol{\lambda})_4$
Zero Masking
input (X)
Output
DOD Model
$(\boldsymbol{\lambda}_{arch})$
$\boldsymbol{\lambda} = [\boldsymbol{\lambda}_{reg}, [3,0,0,5]]$
A
$\mathbf{A} \in \{0,1\}^{(4\times5\times5)}$
PE($\boldsymbol{\lambda}_{arch}$)
$HN(\cdot,\phi)$
$\widehat{\mathbf{W}}_{\phi}(\boldsymbol{\lambda})_1$
$\widehat{\mathbf{W}}_{\phi}(\boldsymbol{\lambda})_2$
$\widehat{\mathbf{W}}_{\phi}(\boldsymbol{\lambda})_3$
$\widehat{\mathbf{W}}_{\phi}(\boldsymbol{\lambda})_4$
input (X)
No-op
No-op
Output
DOD Model

Figure 2: Illustration of the proposed HN. (Top) HN generates weights for a 4-layer AE, with layer widths equal to $[4,2,4,5]$. Weights $\widehat{\mathbf{W}}_{\boldsymbol{\phi}}$ is fed into the DOD model, while hidden layers' dimensions are shrunk by the masking A. (Bottom) HN generates weights for a 2-layer AE, with layer widths equal to $[3,5]$. $\boldsymbol{\lambda}_{arch}$ is padded as $[3,0,0,5]$, and the architecture masking at the second and third layer are set to all zeros. When $\widehat{\mathbf{W}}_{\boldsymbol{\phi}}$ is fed into the DOD model, zero masking enables the "No Operation" (No-op), in effect shrinking the DOD model from 4 layers to 2 layers.

differ (smaller kernels would result in larger output size); therefore, we need to guarantee the spatial size by similarity padding zeros around the input of that convolutional layer. The padding is similar to how we construct the architecture masking $\mathbf{A}$ and similar to the padding approach discussed in [38] .

Diverse Weight Generation. While HN is a universal function approximator in theory, it may not generalize well to offer good approximations for many unseen architectures, especially given that the number of $\boldsymbol{\lambda}$'s during training is limited. When there is only little variation between two inputs, the HN provides more similar weights, since the weights are generated from the same HN where implicit weight sharing occurs.

We employ two ideas toward enabling the HN to generate more expressive weights in response to changes in $\boldsymbol{\lambda}_{arch}$. First is to inject more variation within its input space where, instead of directly feeding in $\boldsymbol{\lambda}_{arch}$, we input the positional encoding of each element in $\boldsymbol{\lambda}_{arch}$. Positional encoding [36] transforms each scalar element into a vector embedding, which encodes more granular information, especially when $\boldsymbol{\lambda}_{arch}$ contains zeros representing a shallower sub-architecture. Second idea is to employ a scheduled training strategy of the HN as it produces weights for both shallow and deep architectures. During HN training, we train with $\boldsymbol{\lambda}$ associated with deeper architectures first, and later, $\boldsymbol{\lambda}$ for shallower architectures are trained jointly with deeper architectures. Our scheduled training alleviates the problem of imbalanced weight sharing, where weights associated with shallower layers are updated more frequently as those are used by more number of architectures. Fig. 3 illustrates how the training losses change for individual architectures during the HN's scheduled training.

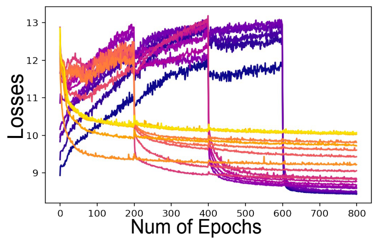


Figure 3: Loss of individual models during scheduled training. Lighter colors depict loss curves of deeper architectures, which enter training early. Over epochs loss is minimized for all models collectively.

Batchwise Training. Like other NNs, HN allows for several inputs $\{\boldsymbol{\lambda}_j\}_{j=1}^m$ synchronously and outputs $\{\widehat{\mathbf{W}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda}_j)\}_{j=1}^m$. To speed up training, we batch the input at each forward step with a set of different architectures and regularization HP configurations. Assuming $\{\boldsymbol{\lambda}_j\}_{j=1}^m = \{[\boldsymbol{\lambda}_{arch,j}, \boldsymbol{\lambda}_{reg,j}]\}_{j=1}^m \subset \boldsymbol{\Lambda}$ are the set of sampled HP configurations from the model space, given training points $\mathbf{X}$, the HN loss for one pass is calculated as:

$$\mathcal{L}_{\text{hn}} = \sum_{\mathbf{x}\in\mathbf{X}} \sum_{j=1}^{m} \mathcal{L}_{\text{trn}}\left(\widehat{\mathbf{W}}_{\boldsymbol{\phi}}([\boldsymbol{\lambda}_{arch,j}, \boldsymbol{\lambda}_{reg,j}]), \mathbf{x}\right) \tag{5}$$

where the training loss $\mathcal{L}_{trn}$ is the same loss as that of the DOD model of interest; e.g. reconstruction loss for autoencoder-based OD models, one-class losses [32], or regularized reconstruction loss [45], etc. We feed the HN-generated weights (instead of learning the actual weights) as well as the training data $\mathbf{X}$ to the DOD model M, which then outputs the outlier scores $\widehat{O}_j := M(\mathbf{X}; \widehat{\mathbf{W}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda}_j))$. The outlier scores are used to compute the training loss $\mathcal{L}_{trn}$, as well as in training our proxy validation function as described next. (See Fig. 1) During training, the gradients can further propagate through the generated weights to update the HN parameters ϕ.

In summary, our HN mimics fast DOD model building across different HP configurations. This offers two advantages: (*i*) training many different HPs jointly in meta-training and (*ii*) fast DOD model parameter generation during online model search. Notably, our HN can tune a wider range of HPs including model architecture, and as shown in §5.3, provides superior results to only tuning $\boldsymbol{\lambda}_{reg}$.

3.2 Validation without Supervision via Meta-learning

Given the lack of ground truth labels on a new dataset, model selection via supervision is not feasible. Instead, we consider transferring supervision from historical datasets through meta-learning, enabling model performance evaluation on the new dataset. Meta-learning uses a collection of historical tasks $\mathcal{D}_{\text{train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_N\}$ that contain ground-truth labels, i.e. $\{\mathcal{D}_i = (\mathbf{X}_i, \mathbf{y}_i)\}_{i=1}^N$. For many OD settings, such historical datasets are obtainable. For example, for a new disease diagnosis, many previous medical records and medical images are available for normal vs. abnormal health conditions, which can be utilized as historical datasets.

Given a DOD algorithm M for UDOMS, we let M_j denote the DOD model with a specific HP setting $\boldsymbol{\lambda}_j$ from the set $\{\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_m\} \subset \boldsymbol{\Lambda}$. HyPer uses $\mathcal{D}_{\text{train}}$ to compute two quantities. First, we obtain *historical outlier scores* of each M_j with HP setting $\boldsymbol{\lambda}_j$ on each $\mathcal{D}_i = (\mathbf{X}_i, \mathbf{y}_i) \in \mathcal{D}_{\text{train}}$. Let $O_{i,j} := M_j(\mathbf{X}_i, \mathbf{W}(\boldsymbol{\lambda}_j))$ denote the output outlier scores of M_j trained with a specific HP configuration $\boldsymbol{\lambda}_j$ for the data points $\mathbf{X}_i$ in dataset $\mathcal{D}_i$, where $\mathbf{W}(\boldsymbol{\lambda}_j)$ are M_j's estimated (i.e. HN-generated) weights for $\boldsymbol{\lambda}_j$. Second, we can calculate the *historical performance matrix* $\mathbf{P} \in \mathbb{R}^{N\times m}$, where $\mathbf{P}_{i,j} := \text{perf}(O_{i,j}, \mathbf{y}_i)$ denotes M_j's detection performance (e.g. AUROC) on dataset $\mathcal{D}_i$, evaluated based on $\mathbf{y}_i$.

With the historical outlier scores and the performance matrix in hand, we train a proxy validator called f_{val}, which provides us with performance estimation $\widehat{P}$ when we encounter a new dataset and no label is given. As shown in Fig. 1 (right), the high-level idea of f_{val} is to learn a mapping from data and model characteristics (e.g., distribution of outlier scores) to the corresponding OD performance across N historical datasets and m models (with different HP configurations). Since it is costly to train all these OD models individually on all datasets from scratch, we utilize our previously proposed HN only *once per dataset* across m different HP configurations, which generates the weights and outlier scores for all models. With the hypernetwork and meta-learning from historical dataset introduced, we present the full framework of HyPer in §4 and the training details of f_{val} in §4.1.

4 HyPer Framework for UDOMS

HyPer consists of two phases (see Fig. 1): (§4.1) offline meta-training over the historical datasets, and (§4.2) online model selection for a given test dataset. In the offline phase, we train the proxy validator f_{val}, which allows us to predict model performance on the test dataset without relying on labels. During online model selection, we alternate between training our HN to efficiently generate model weights for varying HPs around a local neighborhood, and refining the best HPs at the current iteration based on f_{val}'s predictions for many locally sampled HPs. We present the details as follows.

4.1 Offline Training on Historical Datasets

In HyPer, we train a proxy validator f_{val} across historical datasets, so that we can predict the performance of an OD model on the test dataset. f_{val} maps HP configuration ($\boldsymbol{\lambda}$), data embedding, and model embedding onto the corresponding model performance across historical datasets. The goal is to predict detection performance solely based on the characteristics of the input data and the trained model, along with the HP values. We create the data embedding and model embedding as described below.

Data Embeddings. Existing work [43] captures the data characteristics of an OD dataset via extracting meta-features, such as the number of samples and features, to describe a dataset. Although simple and intuitive, meta-features primarily focus on general data characteristics with a heuristic extraction process, and are insufficient in model selection [44]. In this work, we design a principled approach to capture dataset characteristics. First, the datasets may have different feature and sample sizes, which makes it challenging to learn dataset embeddings. To address this, we employ feature hashing [39], $\psi(\cdot)$, to project each dataset to a k-dimensional unified feature space. To ensure sufficient expressiveness, the projection dimension should not be too small ($k = 256$ in our experiments). Subsequently, we train a cross-dataset feature extractor $h(\cdot)$, a fully connected neural network, trained with historical datasets' labels, to learn the mapping from hashed samples to the corresponding outlier labels. i.e. $h : \psi(\mathbf{X}_i) \mapsto \mathbf{y}_i$ for the i-th dataset. Training over datasets, the latent representations by $h(\cdot)$ are expected to capture the outlying characteristics of datasets. Finally, we use max-pooling to aggregate sample-wise representations into dataset-wise embeddings, denoted by $\text{pool}\{h(\psi(\mathbf{X}_i))\}$.

Model Embeddings. In addition to data embeddings, we need model embeddings that change along with the varying hyperparameter settings to train an effective proxy validator. Here we use the historical outlier scores and historical performance matrix, as presented in §3.2, to learn a neural network $g(\cdot)$ that generates the mapping from the outlier scores onto detection performance, i.e. $g : O_{i,j} \mapsto \mathbf{P}_{i,j}$. To handle size variability of outlier scores (due to sample size differences across datasets) as well as to remain agnostic to permutations of outlier scores within a dataset, we employ the DeepSet architecture [40] for $g(\cdot)$, and use the pooling layer's output as the model embedding, denoted by $\text{pool}\{g(O_{i,j})\}$.

Effective and Efficient f_{val} Training. By incorporating the aforementioned components, we propose the proxy validator f_{val}, which tries to learn the following mapping:

$$f_{\text{val}} \; : \; \underbrace{\boldsymbol{\lambda}_j}_{\text{HPs}}, \; \underbrace{\text{pool}\{h(\psi(\mathbf{X}_i))\}}_{\text{data embed.}}, \; \underbrace{\text{pool}\{g(O_{i,j})\}}_{\text{model embed.}} \; \mapsto \; \mathbf{P}_{i,j}$$
$$i \in \{1, \ldots, N\}, \; j \in \{1, \ldots, m\} \qquad (6)$$

We train f_{val} with lightGBM [13] (one may use any regressor), across N historical datasets and m models with varying HP configurations. The loss function is the squared error between prediction $\widehat{\mathbf{P}}$ and $\mathbf{P}$: $\mathcal{L}_f = \sum_{i=1}^{N} \sum_{j=1}^{m} \|\widehat{\mathbf{P}}_{i,j} - \mathbf{P}_{i,j}\|^2$.

By considering both data and model embeddings in Eq. (6), f_{val} predicts performance more effectively compared to existing works that solely rely on HP values and dataset meta-features [43].

Notice that obtaining the model embeddings in Eq. (6) that rely on the outlier scores $O_{i,j}$ requires training the DOD model for each dataset $\mathcal{D}_i$ and each HP configuration $\boldsymbol{\lambda}_j$, which can be computationally expensive. To speed up this process for meta-training, we use HN-generated weights rather than training these individual DOD models from scratch, to obtain $\widehat{O}_{i,j} := M_j(\mathbf{X}_i; \widehat{\mathbf{W}}_{\boldsymbol{\phi}}^{(i)}(\boldsymbol{\lambda}_j))$, where $\widehat{\mathbf{W}}_{\boldsymbol{\phi}}^{(i)}(\boldsymbol{\lambda}_j)$ denotes model M_j's weights for HP configuration $\boldsymbol{\lambda}_j$, as generated by our HN trained on $\mathcal{D}_i$.

At test time, the proxy validator f_{val} provides performance evaluation by taking in the new dataset's embedding and a DOD model's embedding with a HP configuration, *without* requiring the ground truth labels. In this way, we are able to leverage the benefits of meta-learning from historical datasets, and estimate the performance of a specific DOD model on a new dataset, when the new dataset contains no labels.

4.2 Online Model Selection

Model selection via proxy validator. With our meta-trained f_{val} at hand, given a test dataset $\mathbf{X}_{\text{test}}$, a simple model selection strategy would be to train many DOD models with different randomly sampled HPs on $\mathbf{X}_{\text{test}}$ to obtain outlier scores, and then select the one with the highest predicted performance by f_{val}.

However, training OD models from scratch for each HP can be computationally expensive. To speed this up, we also train a HN on $\mathbf{X}_{\text{test}}$ and subsequently obtain the outlier scores $\widehat{O}_{\text{test}}$ from the DOD model with HN-generated weights for randomly sampled HPs. We select the best HP configuration according to Eq. (7).

$$\underset{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}}{\text{argmax}} \; f_{\text{val}}(\mathbf{X}_{\text{test}}, \boldsymbol{\lambda}, \widehat{O}_{\text{test},\boldsymbol{\lambda}}) \qquad (7)$$

Moreover, we propose to iteratively train our HN over locally selected HPs, since training a "global" HN to generate weights across the entire $\boldsymbol{\Lambda}$ and over unseen $\boldsymbol{\lambda}$ is a challenging task especially for large model spaces [22], which can impact the quality of the generated weights and subsequently affect the overall performance of the selected model. Therefore, we propose two stages of alternate updating. One stage trains the HN according to a neighborhood of sampled HP configurations around the current best HP, while the other stage applies the generated weights from HN to obtain outlier scores, and subsequently find the new best HP configuration through the performance proxy validator.

Training local HN iteratively and adaptively. We design HyPer to jointly optimize the HPs $\boldsymbol{\lambda}$ and the (*local*) HN parameters $\boldsymbol{\phi}$ in an alternating fashion; as shown in Fig. 1 (bottom). Algorithm 1 provides the step-by-step outline of the process. Over iterations, it alternates between two stages:

(S1) **HN training** that updates HN parameters $\boldsymbol{\phi}$ to approximate the best-response in a local neighborhood around the current hyperparameters $\boldsymbol{\lambda}_{\text{curr}}$ via $\mathcal{L}_{\text{hn}}$ (Lines 4–8), and

(S2) **HP optimization** that updates $\boldsymbol{\lambda}_{\text{curr}}$ in a gradient-free fashion by estimating detection performance through f_{val} of a large set of candidate $\boldsymbol{\lambda}$'s sampled from the same neighborhood, using the corresponding approximate best-response, i.e. the HN-generated weights, $\widehat{\mathbf{W}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda})$ (Line 9).

To dynamically control the sampling range around $\boldsymbol{\lambda}_{\text{curr}}$, we use a factorized Gaussian with standard deviation $\boldsymbol{\sigma}$ to generate

Algorithm 1 HyPer: Online Model Selection

```
Input: test dataset $\mathcal{D}_{\text{test}} = (\mathbf{X}_{\text{test}}, \emptyset)$, HN parameters $\boldsymbol{\phi}$, HN learning rate $\alpha$, HN loss $\mathcal{L}_{\text{hn}}(\cdot)$, proxy validator $f_{\text{val}}$, HN (re-)training epochs $T$, validation objective $\mathcal{G}(\cdot)$ in Eq. (9), patience $p$
Output: optimized HP configuration $\boldsymbol{\lambda}^*$ for the test dataset

1: Initialize $\boldsymbol{\lambda}_{\text{curr}}$ and $\boldsymbol{\sigma}_{\text{curr}}$
2: while patience criterion $p$ is not met do
3:     Sample set $\mathcal{S} := \emptyset$
4:     for $t = 1, \ldots, T$ do
5:         $\boldsymbol{\epsilon}_t \sim p(\boldsymbol{\epsilon}|\boldsymbol{\sigma}_{\text{curr}})$    ▶ local sampling range around $\boldsymbol{\lambda}_{\text{curr}}$
6:         $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \alpha \frac{\partial}{\partial \boldsymbol{\phi}} \mathcal{L}_{\text{hn}}(\boldsymbol{\lambda}_{\text{curr}} + \boldsymbol{\epsilon}_t, \widehat{\mathbf{W}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda}_{\text{curr}} + \boldsymbol{\epsilon}_t))$
7:         $\mathcal{S} := \mathcal{S} \cup (\boldsymbol{\lambda}_{\text{curr}} + \boldsymbol{\epsilon}_t)$    ▶ save locally sampled HPs
8:     end for
9:     $\boldsymbol{\lambda}_{\text{curr}} \leftarrow \text{argmax}_{\boldsymbol{\lambda} \in \mathcal{S}}\ \mathcal{G}(\boldsymbol{\lambda}, \boldsymbol{\sigma}_{\text{curr}}; \boldsymbol{\phi})$    ▶ Eq. (9)
10:    $\boldsymbol{\sigma}_{\text{curr}} \leftarrow \text{argmax}_{\boldsymbol{\sigma}}\ \mathcal{G}(\boldsymbol{\lambda}_{\text{curr}}, \boldsymbol{\sigma}; \boldsymbol{\phi})$
11: end while
12: return $\boldsymbol{\lambda}^* \approx \text{argmax}_{\boldsymbol{\lambda} \in \mathcal{S}}\ f_{\text{val}}(\mathbf{X}_{\text{test}}, \boldsymbol{\lambda}, \widehat{O}_{\text{test},\boldsymbol{\lambda}})$    ▶ Eq. (10)
```

local HP perturbations $p(\boldsymbol{\epsilon}|\boldsymbol{\sigma})$. We initialize $\boldsymbol{\sigma}$ to be a scale factor vector, each value is within $\mathbb{R}^+$, and dynamically change the value of $\boldsymbol{\sigma}$, which becomes $\boldsymbol{\sigma}_{\text{curr}}$ to control the radius of sampling neighborhood. $\boldsymbol{\sigma}_{\text{curr}}$ is used in (S1) for sampling local HPs and is then updated in (S2) at each iteration (Line 10).

Updating $\boldsymbol{\lambda}_{\text{curr}}$ and $\boldsymbol{\sigma}_{\text{curr}}$. HyPer iteratively explores promising HPs and the corresponding sampling range. To update $\boldsymbol{\lambda}_{\text{curr}}$ and the sampling factor $\boldsymbol{\sigma}_{\text{curr}}$, we maximize:

$$\underbrace{\mathbb{E}_{\boldsymbol{\epsilon}\sim p(\boldsymbol{\epsilon}|\boldsymbol{\sigma})}\left[f_{\text{val}}(\mathbf{X}_{\text{test}}, \boldsymbol{\lambda}+\boldsymbol{\epsilon}, \widehat{\mathbf{W}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda}+\boldsymbol{\epsilon}))\right]}_{\text{update } \boldsymbol{\lambda}_{\text{curr}} \text{ to a better model/HPs w/ high expected performance}} + \underbrace{\tau\, \mathbb{H}(p(\boldsymbol{\epsilon}|\boldsymbol{\sigma}))}_{\text{sampling range around } \boldsymbol{\lambda}_{\text{curr}}} . \quad (8)$$

The objective consists of two terms. The first term emphasizes selecting the next model/HP configuration with high expected performance, aiming to improve the overall model performance. The second term measures the uncertainty of the sampling factor, quantified with Shannon's entropy $\mathbb{H}$. A higher entropy value indicates a less localized sampling, allowing for more exploration. The objective is to find an HP configuration that can achieve high expected performance, within a reasonably large local region to contain a good model, that is also local enough for the HN to be able to effectively learn the best-response. If the sampling factor $\boldsymbol{\sigma}$ is too small, it limits the exploration of the next HP configuration and training of the HN, potentially missing out on better-performing options. Conversely, if $\boldsymbol{\sigma}$ is too large, it may lead to inaccuracies in the HN's generated weights, compromising the accuracy of the first term. The balance factor τ controls the trade-off between the two terms.

We approximate the expectation term in Eq. (8) by the empirical mean of predicted performances through V number of sampled perturbations around $\boldsymbol{\lambda}$. We define our validation objective $\mathcal{G}$ as:

$$\mathcal{G}(\boldsymbol{\lambda}, \boldsymbol{\sigma}; \boldsymbol{\phi}) = \tfrac{1}{V}\textstyle\sum_{i=1}^{V} f_{\text{val}}(\mathbf{X}_{\text{test}}, \boldsymbol{\lambda}+\boldsymbol{\epsilon}_i, \widehat{\mathbf{W}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda}+\boldsymbol{\epsilon}_i)) + \tau\mathbb{H}(p(\boldsymbol{\epsilon}|\boldsymbol{\sigma})) \quad (9)$$

In each iteration of the HP update, we first fix $\boldsymbol{\sigma}_{\text{curr}}$ and find the configuration in $\mathcal{S}$ with the highest value of Eq. (9), where we sample V local configurations around each $\boldsymbol{\lambda} \in \mathcal{S}$, i.e., $\boldsymbol{\lambda} + \boldsymbol{\epsilon}_i|\boldsymbol{\sigma}_{\text{curr}}$ for $i \in 1, \ldots, V$. After $\boldsymbol{\lambda}_{\text{curr}}$ is updated, we fix it and update the sampling factor $\boldsymbol{\sigma}_{\text{curr}}$ also by Eq. (9), using V samples based on each $\boldsymbol{\sigma}$ from a pre-specified range for each HP. To ensure encountering a good HP configuration, we set V to be a large number, e.g. 500. We provide details and pre-specified range in Appx. §A.1.

Table 1: HyPer and baselines for time (in mins) and performance comparison with categorization by whether it selects models (2nd column), uses meta-learning (3rd column), and requires model building at the test time (4th column). Overall, HyPer (with patience $p = 3$) achieves the best detection performances (also see Fig. 4 and 5). Compared to the SOTA ELECT, HyPer has markedly shorter offline and online time.

Method	Model Selection	Meta Learning	Zero shot	Offline Time	Avg. On -line Time	Med. On -line Time	Avg. ROC Rank(↓)
Default	✗	✗	✓	N/A	0	0	0.5954
Random	✗	✗	✓	N/A	0	0	0.5603
MC	✓	✗	✗	N/A	215	277	0.5642
GB	✓	✓	✓	7,461	0	0	0.4668
ISAC	✓	✓	✓	7,466	1	1	0.4181
AS	✓	✓	✓	7,465	1	1	0.5222
MetaOD	✓	✓	✓	7,525	1	1	0.3918
ELECT	✓	✓	✗	7,611	59	71	0.3621
Ours	✓	✓	✗	1,320	14	17	0.2954

Selecting the Best Model/HP $\boldsymbol{\lambda}^*$. We employ f_{val} to choose the best HP $\boldsymbol{\lambda}^*$ among all the locally sampled HPs $\mathcal{S}$ during the last iteration of HN training. Note that HyPer directly uses the HN-generated weights for fast computation, without the need to train any model from scratch for evaluation by f_{val}. With the generated weights $\widehat{\mathbf{W}}_{\phi}(\boldsymbol{\lambda})$, the DOD model produces the corresponding outlier scores, denoted as $\widehat{O}_{\text{test},\boldsymbol{\lambda}} := M_{\boldsymbol{\lambda}}(\mathbf{X}_{\text{test}}; \widehat{\mathbf{W}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda}))$, that allows us to select the best HP configuration within the candidate set by

$$\boldsymbol{\lambda}^* \approx \underset{\boldsymbol{\lambda} \in \mathcal{S}}{\text{argmax}}\ f_{\text{val}}(\mathbf{X}_{\text{test}}, \boldsymbol{\lambda}, \widehat{O}_{\text{test},\boldsymbol{\lambda}}) . \quad (10)$$

Initialization and Convergence. We initialize $\boldsymbol{\lambda}_{\text{curr}}$ and $\boldsymbol{\sigma}_{\text{curr}}$ with the globally best values across historical datasets. We consider HyPer as converged if the highest predicted performance by f_{val} does not improve in p consecutive iterations. A larger p, referred as "patience", requires more iterations to converge yet likely yields better results. Note that p can be decided by cross-validation on historical datasets during meta-training. We present an empirical analysis of initialization and patience in the experiments.

5 Experiments

5.1 Experiment Settings

Benchmark Data. We show HyPer's effectiveness and efficiency with fully connected AutoEncoder (AE) for DOD on tabular data, using a testbed consisting of 34 benchmark datasets from two different public OD repositories; ODDS [30] and DAMI [5] (`Pima` dataset is removed). In addition, we run HyPer with convolutional AE on MNIST and FashinMNIST datasets. We treat one class as the normal class, while downweighting the ratio of another class at 10% as the outlier class. We train and validate HyPer with inliers and outliers from classes [0,5] (30 tasks/datasets in total) and evaluate 8 tasks on [6,9], to avoid data leakage in (meta)training/testing data.

Baselines. For tabular dataset, we include 8 baselines for comparison ranging from simple to state-of-the-art (SOTA); Table 1 provides a conceptual comparison of the baselines. They are organized as (*i*) ***no model selection***: **(1) Default** uses the default HPs used in a popular OD library PyOD [42], **(2) Random** picks an HP/model randomly (we report expected performance); (*ii*): ***model***

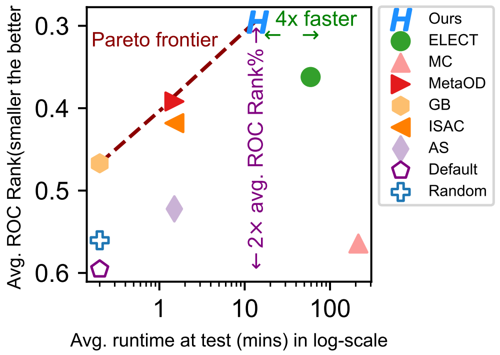


Figure 4: Avg. running time (log-scale) vs. avg. model ROC Rank. Meta-learning methods are depicted with solid markers. Pareto frontier (red dashed line) shows the best methods under different time budgets. HyPer **outperforms all with reasonable computational demand.**

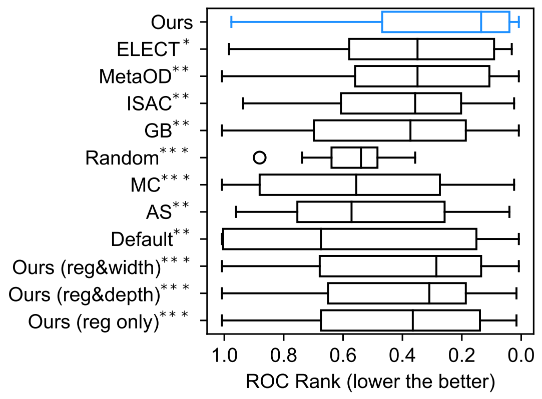


Figure 5: Distribution of ROC Rank across datasets. HyPer **achieves the best performance. Bottom three bars depict** HyPer**'s variants that do not fully tune architecture HPs (for ablation). Paired test results are depicted as significant w/ * at 0.1, ** at 0.01, *** at 0.001. See p-values in Appx. Table A1.**

selection without meta-learning: **(3) MC** [20] leverages consensus; and (*iii*) ***model selection by meta-learning***: **(4) Global Best (GB)** selects the best performing model on the historical datasets on average, and SOTA baselines include **(5) ISAC [12]**, **(6) ARGOSMART (AS) [27]**, **(7) MetaOD** [43], **(8) ELECT** [44]. Baselines (1), (2), and (4)-(7) are zero-shot that do not require model building during model selection. For image dataset, we proivde HyPer's performance in comparison to Random (avg. performance across HPs) and Default (configuration used by DeepSVDD [33]).

Evaluation. For tabular data, we use 5-fold cross-validation to split the train/test datasets; that is, each time we use 28 datasets as the historical datasets to select models on the remaining 7 datasets. For image data, we use 5-fold cross-validation: 24 (out of 30) datasets as the historical datasets to select models on the remaining 6 (meta-train) datasets. We use AUROC to measure detection performance, while it can be substituted with any other measure. We report the raw ROC as well as the normalized ROC *Rank* of an HP/model, ranging from 0 (the best) to 1 (the worst)—i.e., the lower the better.We use the paired Wilcoxon signed rank test [10] across all datasets in the testbed to compare two methods. Full results in Appx. §A.2.

5.2 Experiment Results

Tabular. Fig. 4 shows that HyPer **outperforms all baselines with regard to average ROC Rank on the 35 testbed.**In addition, Fig. 5 provides the full performance distribution across all datasets and shows that HyPer is statistically better than all baselines, including SOTA meta-learning based ELECT and MetaOD. Among the zero-shot baselines, Default and Random perform significantly poorly while the meta-learning based GB leads to comparably higher performance. Same as previous study [20], the internal consensus-based MC can be no better than Random.

HN-powered efficiency enables HyPer **to search more broadly**. Fig. 4 and Appx. Table 1 show that HyPer offers significant speed up over the SOTA method ELECT, with an average offline training speed-up of 5.77× and a model selection speed-up of 4.21×. Unlike ELECT, which requires building OD models from scratch during both offline and online phases, HyPer leverages the HN-generated weights to avoid costly model training for each candidate HP.

Meanwhile, HyPer can also afford a broader range of HP configurations thanks to the lower model building cost by HN. This capability contributes to the effectiveness of HyPer, which brings 7% avg. ROC Rank ↑ over ELECT.

Meta-learning methods achieve the best performance at different budgets. Fig. 4 and Appx. Table 1 show that the best performers at different time budgets are global best (GB), MetaOD, and HyPer, which are all on the Pareto frontier. In contrast, simple no-model-selection approaches, i.e., Default and Random, are typically the lowest performing methods. Specifically, HyPer achieves a significant 2× avg. ROC Rank improvement over the default HP in PyOD [42]), a widely used open-source OD library. Although meta-learning entails additional (offline) training time, it can be amortized across multiple future tasks in the long run.

Image. Table 2 shows that HyPer is outperforming both Random selection and Default LeNet AutoEncoder model, as it is able to effectively learn from historical data to tune HPs. In addition, we evaluate HyPer 's performance across datasets by online training with 5 FashionMNIST tasks, with the same search space of HPs and same offline-training solely on MNIST anomaly detection tasks. Table 3 shows that HyPer outperforms Random on average,but not necessarily on all datasets. It may be due to the fact that MNIST (used for meta-learning) and FashionMNIST (test tasks) are from different distributions and share less similarities for effective transfer through meta-learning.

Table 2: Test AUROC of HyPer **in comparison to Random and Default HP configurations. Overall,** HyPer **shows higher AUROC than Random and Default.**

Inlier/Outlier Class	Random	Default	HyPer
Inlier: 6 Outlier: 9	0.8343	0.8015	**0.9463**
Inlier: 9 Outlier: 7	0.6358	0.6409	**0.7761**
Inlier: 8 Outlier: 6	0.7152	0.6358	**0.8490**
Inlier: 6 Outlier: 8	0.7827	0.8276	**0.8365**
Inlier: 6 Outlier: 7	0.7787	0.7968	**0.9179**
Inlier: 7 Outlier: 6	0.8720	0.8330	**0.8909**
Inlier: 9 Outlier: 6	0.7642	0.7371	**0.9098**
Inlier: 7 Outlier: 9	0.4073	**0.5769**	0.4613

Table 3: Test AUROC of HyPer and Random, evaluated on FashionMNIST Datasets. HyPer, solely trained on MNIST, does not provide competitive performances in all cases.

Inlier/Outlier Class	Random	HyPer
Inlier: T-shirt Outlier: Trouser	0.6004	**0.6132**
Inlier: Trouser Outlier: Pullover	0.8902	**0.9878**
Inlier: Dress Outlier: Coat	**0.8715**	0.8456
Inlier: Sandal Outlier: Shirt	**0.9024**	0.8752
Inlier: Sneaker Outlier: Bag	**0.9457**	0.9033

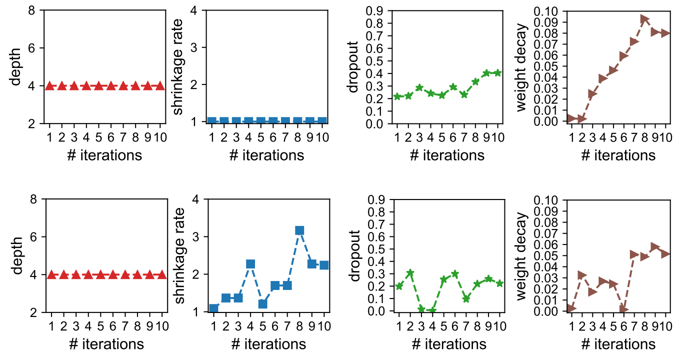


Figure 6: Trace of HP changes over iterations on spamspace: (top) tuning regularization HPs only; (bottom) tuning both regularization and architectural HPs (ours). When arch. is fixed, reg. HPs incur more magnitude changes and reach larger values to adjust model complexity. HyPer tunes complexity more flexibly by also accommodating arch. HPs.

5.3 Ablation Studies

Benefit of Tuning Architectural HPs via HN. HyPer tackles the challenging task of accommodating architectural HPs besides regularization HPs. Through ablations, we study the benefit of our novel HN design, as presented in §3.1, which can generate DOD model weights in response to changes in architecture. Bottom three bars of Fig. 5 show the performances of three HyPer variants across datasets. The proposed HyPer (with median ROC Rank = 0.1349) outperforms all these variants significantly (with p<0.001), namely, *only tuning regularization and width* (median ROC Rank = 0.2857), *only tuning regularization and depth* (median ROC Rank = 0.3095), and *only tuning regularization* (median ROC Rank = 0.3650). By extending its search for both neural network depth *and* width, HyPer explores a larger model space that helps find better-performing model configurations.

HP Schedules over Iterations. In Fig. 6, we more closely analyze how HPs change over iterations on spamspace, comparing between (top) only tuning reg. HPs while fixing model depth and width (i.e., shrinkage rate) and (bottom) using HyPer to tune all HPs including both reg. and architectural HPs. Bottom figures show that depth remains fixed at 4, shrinkage rate increases from 1 to 2.25 (i.e., width gets reduced), dropout to 0.2, and weight decay to 0.05—overall model capacity is reduced relative to initialization. In contrast, top figures show that, when model depth and width are fixed, regularization HPs compensate more to adjust the model capacity, with a larger dropout rate at 0.4 and larger weight decay at 0.08, achieving ROC rank 0.3227 in contrast to HyPer's 0.0555. This comparison showcases the merit of HyPer which adjusts model complexity more flexibly by accommodating a larger model space.

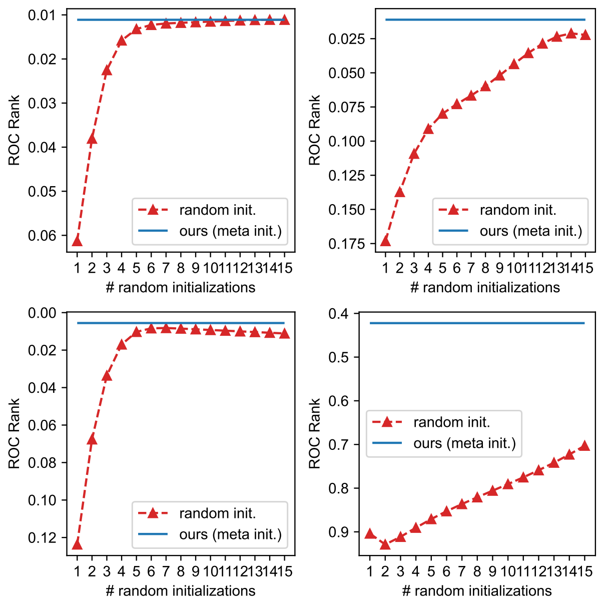


Figure 7: Comparison of ROC Rank (lower is better) of HyPer with meta-initialization (in blue) with increasing numbers of randomly initialized HNs, on ODDS_wine (upper left), WDBC (upper right), HeartDisease (lower left) and Ionosphere (lower right). It needs 9 randomly initialized HNs to achieve the same performance as HyPer on ODDS_wine . In general, HyPer finds a good model with much less running time.

Effect of Meta-initialization. In Fig. 7, we demonstrate the effectiveness of meta-initialization by comparing it with random initialization on four datasets. In addition to utilizing meta-initialization, one could run HyPer multiple times with randomly initialized HPs and select the best model based on f_{val}. To simulate this scenario, we vary the number of random initializations (x-axis) and record all the f_{val} values along with the corresponding ROC Rank. For each dataset, we select the best model based on f_{val} across *all* trials. We increase the number of random trials from 1 to 15, where the highest f_{val} value among the 15 random initialized trials is chosen as the best model. Meta-initialization is indeed a strong starting point for HyPer's HP tuning. For example, on the ODDS_wine dataset, it requires 9 randomly initialized HNs to attain the same performance as our approach with meta-initialization, showing a 9-fold increase in the time required for online selection. In other cases, training 15 randomly initialized HNs fails to achieve the same performance as meta-initialization, further validating its advantages.

Effect of Patience. The convergence criterion for HyPer is based on the highest predicted performance by f_{val} remaining unchanged for p consecutive iterations ("patience"). As illustrated in Fig. 8, increasing the value of p allows for more exploration and

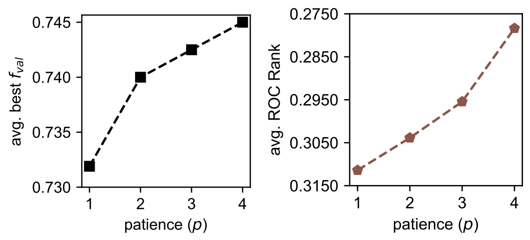


Figure 8: Analysis of the effect of patience p: (left) avg. f_{val} value change when increasing p from 1 to 4; (right) avg. ROC Rank (lower is better) with increasing p. Larger p leads to more exploration and tends to offer better performance.

potentially better performance. However, this also prolongs the convergence time. In our experiments, we set $p = 3$ to balance performance and runtime. The specific value of p can be determined through cross-validation over the historical datasets.

5.4 Limitation

To analyze HyPer 's performance with respect to train/test datasets, we compare HyPer 's predictions to the test dataset's similarity to the training datasets. We first adapt the code from MetaOD's feature extractor[1] and extract features that represent the underlying data distribution, including mean, standard deviation, kurtosis, sparsity, skewness, and etc. Since each dataset is now represented as a vector of meta-features, we are able to measure the pairwise cosine similarity between datasets. For each dataset, we then calculate the average cosine similarity to the training datasets.

Table 4 shows the Top-5 datasets where HyPer has the most AUROC performance differences to the Top-1 baseline (listed in Table A3), and the dataset's average cosine similarity to the training datasets. We observe that HyPer 's performance can be subpar when the test dataset has a small cosine similarity to the training datasets, since all the 5 datasets have smaller cosine similarity than the average pairwise dataset similarity. We can further conclude that one working assumption of HyPer is that test dataset has a similar data distribution to at least a few of the training datasets.

Table 4: Average Cosine Similarity to Training Dataset vs. HyPer's AUROC Difference. HyPer performs worse when test dataset has small consine similarity to training data.

Dataset	Avg. Cos Sim.	AUROC Diff. (Rank)
DAMI_Wilt	0.4570	0.1371 (9)
ODDS_vertebral	0.2610	0.1262 (7)
DAMI_Annthyroid	0.1955	0.1236 (8)
DAMI_Glass	0.2451	0.0525 (7)
ODDS_annthyroid	0.2446	0.0339 (5)
Avg. Pairwise Sim.	**0.5194**	-

6 Related Work

Supervised Model Selection. Supervised model selection leverages hold-out data with labels. Randomized [2], bandit-based [16], and Bayesian optimization (BO) techniques [35] are various leading approaches. Self-tuning networks (STN) [22] utilizes validation data to alternatively update the HPs in the HP space along with the corresponding model weights. Under the context of OD, recent work include AutoOD [17] that focuses on neural architecture search, as well as PyODDS [18] and TODS [15] for model selection, all of which rely on hold-out labeled data. Clearly, these supervised approaches do not apply to UDOMS.

Unsupervised Model Selection. To choose OD models in an unsupervised fashion, one approach is to design unsupervised internal evaluation metrics [9, 23, 26] that solely depend on input features, outlier scores, and/or the learned model parameters. However, a recent large-scale study showed that most internal metrics have limited performance in unsupervised OD model selection [20]. More recent solutions leverage meta-learning that selects the model for a new dataset by using the information on similar historical datasets—SOTA methods include MetaOD [43] and ELECT [44]. Their key bottleneck is efficiently training the candidate models with different HPs, which is addressed in our work.

Hypernetworks. Hypernetworks (HN) have been primarily used for parameter-efficient training of large models with diverse architectures [3, 14, 22, 41] as well as generating weights for diverse learning tasks [29, 37]. HN generates weights (i.e. parameters) for another larger network (called the main network) [11]. As such, one can think of the HN as a model compression tool for training, one that requires fewer learnable parameters. Going back in history, hypernetworks can be seen as the birth-child of the "fast-weights" concept by Schmidhuber [34], where one network produces context-dependent weight changes for another network. The context, in our as well as several other work [3, 22], is the hyperparameters (HPs). That is, we train a HN model that takes (encoding of) the HPs of the (main) DOD model as input, and produces HP-dependent weight changes for the DOD model that we aim to tune. Training a single HN that can generate weights for the (main) DOD model for varying HPs can effectively bypass the cost of fully-training those candidate models from scratch.

7 Conclusion

We introduced HyPer, a new framework for unsupervised deep outlier model selection. HyPer tackles two fundamental challenges that arise in this setting: validation in the absence of supervision and efficient search of the large model space. To that end, it employs meta-learning to train a proxy validation function on historical datasets to effectively predict model performance on a new task without labels. To speed up search, it utilizes a novel hypernetwork design that generates weights for the detection model with varying HPs including model architecture, achieving significant efficiency gains over individually training the candidate models. Extensive experiments on a large testbed with 35 benchmark datasets showed that HyPer significantly outperforms 8 simple to SOTA baselines. We expect that our work will help practitioners use existing deep OD models more effectively as well as foster further work on unsupervised model selection in the era of deep learning.

[1]https://github.com/yzhao062/MetaOD/blob/master/metaod/models/gen_meta_features.py

References

[1] Charu C. Aggarwal and Saket Sathe. 2015. Theoretical Foundations and Algorithms for Outlier Ensembles. *SIGKDD Explor.* 17, 1 (2015), 24–47.

[2] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* 13 (2012), 281–305. http://dblp.uni-trier.de/db/journals/jmlr/jmlr13.html#BergstraB12

[3] Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston. 2018. SMASH: One-Shot Model Architecture Search through HyperNetworks.. In *ICLR (Poster)*. OpenReview.net.

[4] Guilherme Oliveira Campos, Arthur Zimek, Jörg Sander, Ricardo J. G. B. Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E. Houle. 2016. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Min. Knowl. Discov.* 30, 4 (2016), 891–927.

[5] Guilherme Oliveira Campos, Arthur Zimek, Jörg Sander, Ricardo J. G. B. Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E. Houle. 2016. On the evaluation of unsupervised outlier detection. *DAMI* 30, 4 (2016), 891–927.

[6] Xueying Ding, Lingxiao Zhao, and Leman Akoglu. 2022. Hyperparameter Sensitivity in Deep Outlier Detection: Analysis and a Scalable Hyper-Ensemble Solution. In *Advances in Neural Information Processing Systems*, Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (Eds.).

[7] Sunny Duan, Loic Matthey, Andre Saraiva, Nick Watters, Christopher Burgess, Alexander Lerchner, and Irina Higgins. 2020. Unsupervised Model Selection for Variational Disentangled Representation Learning.. In *ICLR*. OpenReview.net. http://dblp.uni-trier.de/db/conf/iclr/iclr2020.html#DuanMSWBLH20

[8] Matthias Feurer and Frank Hutter. 2019. Hyperparameter optimization. *Automated machine learning: Methods, systems, challenges* (2019), 3–33.

[9] Nicolas Goix. 2016. How to Evaluate the Quality of Unsupervised Anomaly Detection Algorithms? *CoRR* abs/1607.01152 (2016). http://dblp.uni-trier.de/db/journals/corr/corr1607.html#Goix16

[10] David J. Groggel. 2000. Practical Nonparametric Statistics. *Technometrics* 42, 3 (2000), 317–318.

[11] David Ha, Andrew M. Dai, and Quoc V. Le. 2017. HyperNetworks.. In *ICLR (Poster)*. OpenReview.net.

[12] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. 2010. ISAC - Instance-Specific Algorithm Configuration.. In *ECAI*, Vol. 215. 751–756.

[13] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).

[14] Boris Knyazev, Michal Drozdzal, Graham W Taylor, and Adriana Romero Soriano. 2021. Parameter prediction for unseen deep architectures. *Advances in Neural Information Processing Systems* 34 (2021), 29433–29448.

[15] Kwei-Herng Lai, Daochen Zha, Guanchu Wang, Junjie Xu, Yue Zhao, Devesh Kumar, Yile Chen, Purav Zumkhawaka, Minyang Wan, Diego Martinez, and Xia Hu. 2021. TODS: An Automated Time Series Outlier Detection System. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021.* AAAI Press, 16060–16062. https://ojs.aaai.org/index.php/AAAI/article/view/18012

[16] Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *J. Mach. Learn. Res.* 18 (2017), 185:1–185:52. http://dblp.uni-trier.de/db/journals/jmlr/jmlr18.html#LiJDRT17

[17] Yuening Li, Zhengzhang Chen, Daochen Zha, Kaixiong Zhou, Haifeng Jin, Haifeng Chen, and Xia Hu. 2021. AutoOD: Neural Architecture Search for Outlier Detection. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021.* IEEE, 2117–2122. https://doi.org/10.1109/ICDE51399.2021.00210

[18] Yuening Li, Daochen Zha, Praveen Kumar Venugopal, Na Zou, and Xia Hu. 2020. PyODDS: An End-to-end Outlier Detection System with Automated Machine Learning. In *Companion of The 2020 Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, Amal El Fallah Seghrouchni, Gita Sukthankar, Tie-Yan Liu, and Maarten van Steen (Eds.). ACM / IW3C2, 153–157. https://doi.org/10.1145/3366424.3383530

[19] Zinan Lin, Kiran Thekumparampil, Giulia Fanti, and Sewoong Oh. 2020. InfoGAN-CR and ModelCentrality: Self-supervised model training and selection for disentangling GANs. In *International Conference on Machine Learning*. PMLR, 6127–6139.

[20] Martin Q Ma, Yue Zhao, Xiaorong Zhang, and Leman Akoglu. 2023. The Need for Unsupervised Outlier Model Selection: A Review and Evaluation of Internal Evaluation Strategies. *ACM SIGKDD Explorations Newsletter* 25, 1 (2023).

[21] Xiaoxiao Ma, Jia Wu, Shan Xue, Jian Yang, Chuan Zhou, Quan Z Sheng, Hui Xiong, and Leman Akoglu. 2021. A comprehensive survey on graph anomaly detection with deep learning. *IEEE Transactions on Knowledge and Data Engineering* (2021).

[22] Matthew MacKay, Paul Vicol, Jonathan Lorraine, David Duvenaud, and Roger B. Grosse. 2019. Self-Tuning Networks: Bilevel Optimization of Hyperparameters using Structured Best-Response Functions.. In *ICLR (Poster)*. OpenReview.net.

[23] Henrique O. Marques, Ricardo J. G. B. Campello, Jörg Sander, and Arthur Zimek. 2020. Internal Evaluation of Unsupervised Outlier Detection. *ACM Trans. Knowl. Discov. Data* 14, 4 (2020), 47:1–47:42. http://dblp.uni-trier.de/db/journals/tkdd/tkdd14.html#MarquesCSZ20

[24] Henrique O. Marques, Ricardo J. G. B. Campello, Arthur Zimek, and Jörg Sander. 2015. On the internal evaluation of unsupervised outlier detection.. In *SSDBM*. ACM, 7:1–7:12. http://dblp.uni-trier.de/db/conf/ssdbm/ssdbm2015.html#MarquesCZS15

[25] Charles H Martin, Tongsu Peng, and Michael W Mahoney. 2021. Predicting trends in the quality of state-of-the-art neural networks without access to training or testing data. *Nature Communications* 12, 1 (2021), 4122.

[26] Van Nguyen, Thanh Nguyen, and Uy Nguyen. 2017. An Evaluation Method for Unsupervised Anomaly Detection Algorithms. *Journal of Computer Science and Cybernetics* 32, 3 (2017), 259–272. https://doi.org/10.15625/1813-9663/32/3/8455

[27] Mladen Nikolic, Filip Maric, and Predrag Janicic. 2013. Simple algorithm portfolio for SAT. *Artif. Intell. Rev.* 40, 4 (2013), 457–465.

[28] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. 2021. Deep learning for anomaly detection: A review. *ACM computing surveys (CSUR)* 54, 2 (2021), 1–38.

[29] Marcin Przewięźlikowski, Przemysław Przybysz, Jacek Tabor, M Zięba, and Przemysław Spurek. 2022. HyperMAML: Few-Shot Adaptation of Deep Models with Hypernetworks. *arXiv preprint arXiv:2205.15745* (2022).

[30] Shebuti Rayana. 2016. ODDS Library.

[31] Lukas Ruff, Jacob R Kauffmann, Robert A Vandermeulen, Grégoire Montavon, Wojciech Samek, Marius Kloft, Thomas G Dietterich, and Klaus-Robert Müller. 2021. A unifying review of deep and shallow anomaly detection. *Proc. IEEE* 109, 5 (2021), 756–795.

[32] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. 2018. Deep One-Class Classification. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 4393–4402. https://proceedings.mlr.press/v80/ruff18a.html

[33] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. 2018. Deep One-Class Classification. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 4393–4402. https://proceedings.mlr.press/v80/ruff18a.html

[34] Jürgen Schmidhuber. 1992. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation* 4, 1 (1992), 131–139.

[35] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. 2016. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proc. IEEE* 104, 1 (2016), 148–175. https://doi.org/10.1109/JPROC.2015.2494218

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[37] Johannes von Oswald, Christian Henning, Benjamin F. Grewe, and João Sacramento. 2020. Continual learning with hypernetworks. In *International Conference on Learning Representations*. https://arxiv.org/abs/1906.00695

[38] Xiaoxing Wang, Chao Xue, Junchi Yan, Xiaokang Yang, Yonggang Hu, and Kewei Sun. 2021. Mergenas: Merge operations into one for differentiable architecture search. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*. 3065–3072.

[39] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*. 1113–1120.

[40] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. 2017. Deep sets. *Advances in neural information processing systems* 30 (2017).

[41] Chris Zhang, Mengye Ren, and Raquel Urtasun. 2019. Graph HyperNetworks for Neural Architecture Search.. In *ICLR (Poster)*. OpenReview.net.

[42] Yue Zhao, Zain Nasrullah, and Zheng Li. 2019. PyOD: A Python Toolbox for Scalable Outlier Detection. *Journal of Machine Learning Research* 20 (2019), 1–7.

[43] Yue Zhao, Ryan Rossi, and Leman Akoglu. 2021. Automatic Unsupervised Outlier Model Selection. In *Thirty-Fifth Conference on Neural Information Processing Systems*.

[44] Yue Zhao, Sean Zhang, and Leman Akoglu. 2022. Toward Unsupervised Outlier Model Selection. In *IEEE International Conference on Data Mining, ICDM*. IEEE, 773–782.

[45] Chong Zhou and Randy C. Paffenroth. 2017. Anomaly Detection with Robust Deep Autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. Association for Computing Machinery, New York, NY, USA, 665–674. https://doi.org/10.1145/3097983.3098052

Appendix

A Additional Experiment Settings and Results

A.1 Algorithm Settings and Baselines

Setting of HN: The HN utilized in the experiments consists of two hidden layers, each containing 200 neurons. It is configured with a learning rate of 1e-4, a dropout rate of 0.2, and a batch size of 512. We find this setting give enough capacity to generate various weights for linearAEs. Because of the meta-learning setting, the hyperparameters of HN can be tested with validation data and test results, on historical data.

Meta-training for $f_{\mathbf{val}}$. Table A2 includes the HP search space for training in fully-connected AE. In the table, compression rate refers to how many of the widths to shrink between two adjacent layers. For example, if the first layer has width of 6, compression_rate equals 2 would gvie the next layer width equal to 3. We also notice that some datasets may have smaller numbers of features. Thus, with the corresponding compression rate, we also have discretized the width to the nearest integer number. Thus, for some datasets, the HP search space will be smaller than 240. In addition, for HPs in Convolutional AE, HyPer conducts a search among the following HPs: Number of encoders is within $[2, 3, 4]$, kernel size is in $[3, 4, 5]$, channels is $[8, 16, 32]$, lr is $[1e^{-4}, 5e^{-4}]$, weight decay is $[0, 1e^{-4}, 1e^{-5}]$, and dropout is $[0, 0.1, 0.2]$.

HN (Re-)Training during the Online Phase: In order to facilitate effective local re-training, we set a training epoch of $T = 100$ for each iteration, indicating the sampling of 100 local HPs for HN retraining. In Eq. (9), we designate the number of sampled HPs and the sampling factor as 500, i.e., $V_\lambda = V_\sigma = 500$.

Specification of $\boldsymbol{\sigma}$: It is noted that for some values of $\boldsymbol{\sigma}$, the sampled $\boldsymbol{\lambda}$ may not be a valid HP configuration. For example, it is not possible to have floating number as number of layers, or it is not practical when dropout is larger than 0.5. We discard the impossible $\boldsymbol{\lambda}$.

Convergence: To achieve favorable performance within a reasonable timeframe, we set the patience value as $p = 3$.

Baselines: We have incorporated 8 baselines, encompassing a spectrum from simple to state-of-the-art (SOTA) approaches. Table 1 offers a comprehensive conceptual comparison of these baselines.

(*i*) ***no model selection***:

(1) **Default** employs the default HPs utilized in the widely-used OD library PyOD [42]. This serves as the default option for practitioners when no additional information is available.
(2) **Random** randomly selects an HP/model (the reported performance represents the expected value obtained by averaging across all DOD models).

(*ii*) ***model selection without meta-learning***:

(3) **MC** [20] utilizes the consensus among a group of DOD models to assess the performance of a model. A model is considered superior if its outputs are closer to the consensus of the group. MC necessitates the construction of a model group during the testing phase. For more details, please refer to a recent survey [21].

(*iii*) ***model selection by meta-learning*** requires first building a corpus of historical datasets on a group of defined DOD models and then selecting the best from the model set at the test time. Although these baselines utilize meta-learning, none of them take advantage of the HN for acceleration.

(4) **Global Best (GB)** selects the best-performing model based on the average performance across historical datasets.
(5) **ISAC [12]** groups historical datasets into clusters and predicts the cluster of the test data, subsequently outputting the best model from the corresponding cluster.
(6) **ARGOSMART (AS) [27]** measures the similarity between the test dataset and all historical datasets, and then outputs the best model from the most similar historical dataset.
(7) **MetaOD** [43] employs matrix factorization to capture both dataset similarity and model similarity, representing one of the state-of-the-art methods for unsupervised OD model selection.
(8) **ELECT** [44] iteratively identifies the best model for the test dataset based on performance similarity to the historical dataset. Unlike the above meta-learning approaches, ELECT requires model building during the testing phase to compute performance-based similarity.

Baseline Model Set. We use the same HP search spaces for baseline models as well as the HN-trained models. Table A2 provides the detailed HP search space for fully connected AE. For Conv AE, the HP search space is: Number of encoders is within $[2, 3, 4]$, kernel size is in $[3, 4, 5]$, channels is $[8, 16, 32]$, lr is $[1e^{-4}, 5e^{-4}]$, weight decay is $[0, 1e^{-4}, 1e^{-5}]$, and dropout is $[0, 0.1, 0.2]$.

A.2 Additional Results

In addition to the distribution plot in Fig. 5, we provide the p-values of Wilcoxon signed rank test between HyPer and baselines in A1. See §5 for the experiment analysis. Full results are in Table A3.

Table A1: Pairwise statistical tests between HyPer and baselines by Wilcoxon signed rank test. HyPer are statistically better than baselines at different significance levels.

Ours	Baseline	p-value
Ours	Default	0.0035
Ours	Random	0.0003
Ours	ISAC	0.0042
Ours	AS	0.0081
Ours	MetaOD	0.0051
Ours	Global Best	0.0021
Ours	MC	0.0008
Ours	ELECT	0.0803
Ours	Ours (reg&width)	0.0001
Ours	Ours (reg&depth)	0.0001
Ours	Ours (reg only)	0.0001

Table A2: Hyperparameter search space for both free-range and HN models. We give the list of HPs as well as the range of the selected HPs.

List of Hyperparameters (HPs)	# HPs
n_layers: [2,4,6,8]	4
compression_rate: [1.0,1.2,1.4,1.6,1.8,2.0,2.2,2.4,2.6,2.8,3.0]	10
dropout: [0.0,0.2,0.4]	3
weight_decay: [0.0,1e-6,1e-5]	3
Total Number:	240

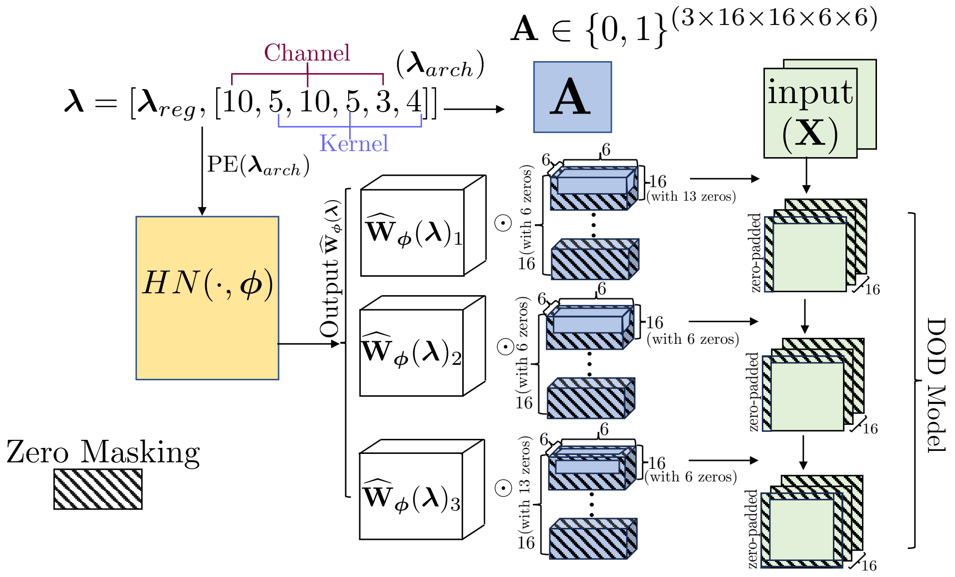

$\mathbf{A} \in \{0,1\}^{(3\times16\times16\times6\times6)}$
Channel
$(\boldsymbol{\lambda}_{arch})$
$\boldsymbol{\lambda} = [\boldsymbol{\lambda}_{reg}, [10, 5, 10, 5, 3, 4]]$
Kernel
A
input (X)
PE($\boldsymbol{\lambda}_{arch}$)
$HN(\cdot, \phi)$
Output $\widehat{\mathbf{W}}_{\phi}(\boldsymbol{\lambda})$
$\widehat{\mathbf{W}}_{\phi}(\boldsymbol{\lambda})_1$
$\widehat{\mathbf{W}}_{\phi}(\boldsymbol{\lambda})_2$
$\widehat{\mathbf{W}}_{\phi}(\boldsymbol{\lambda})_3$
16 (with 13 zeros)
16 (with 6 zeros)
16 (with 6 zeros)
(with 6 zeros) 16
(with 6 zeros) 16
(with 13 zeros) 16
zero-padded
DOD Model
Zero Masking

Figure 9: Illustration of the proposed HN. HN generates weights for a 3-layer convolutional networks , with channels equal to $[10, 10, 3]$**, and kernels equal to** $[5, 5, 4]$**. The HN weights** $\widehat{\mathrm{W}}_{\phi}$ **is of size** $3 \times 16 \times 16 \times 6 \times 6$**, and similarly we construct the same-size architecture masking A. At the first layer, we need to pad A for 1 zero, among the third and fourth dimension (we pad starting from the left and from the top). This will enable us to extend** $\widehat{\mathrm{W}}_{\phi}$ **to a convolutional operation of kernel size** 5**, from fixed kernel size** 6**. To match the padding operation, we also pad the input X along the first and second dimension, with 1. The rest layers follow similairly.**

Table A3: ROC and rank of the evaluated methods. The best method per dataset (row) is highlighted in bold.

Dataset	Default	Random	MC	GB	ISAC	AS	MetaOD	ELECT	Ours
DAMI_Annthyroid	**0.7124 (1)**	0.5972 (6)	0.6123 (3)	0.5929 (7)	0.6018 (5)	0.5873 (9)	0.6050 (4)	0.6148 (2)	0.5888 (8)
DAMI_Cardiotocography	0.7159 (6)	0.7202 (5)	0.7024 (7)	0.7740 (3)	0.7571 (4)	0.6940 (8)	0.6458 (9)	0.7818 (2)	**0.7866 (1)**
DAMI_Glass	**0.7442 (1)**	0.7055 (6)	0.6304 (9)	0.7244 (3)	0.6699 (8)	0.7230 (4)	0.7225 (5)	0.7431 (2)	0.6917 (7)
DAMI_HeartDisease	0.3045 (9)	0.4276 (6)	0.4214 (7)	0.5250 (5)	0.5382 (2)	0.4214 (7)	0.5312 (4)	0.5348 (3)	**0.5926 (1)**
DAMI_PageBlocks	0.8722 (8)	0.9107 (5)	0.9219 (2)	0.9162 (4)	**0.9255 (1)**	0.9002 (6)	0.6247 (9)	0.8791 (7)	0.9215 (3)
DAMI_PenDigits	0.3837 (9)	0.5248 (6)	0.5422 (5)	0.5491 (4)	0.5069 (8)	**0.6953 (1)**	0.6278 (3)	0.5084 (7)	0.6792 (2)
DAMI_Shuttle	0.6453 (8)	0.9462 (2)	0.9400 (5)	0.9342 (7)	0.9436 (3)	**0.9530 (1)**	0.5525 (9)	0.9405 (4)	0.9391 (6)
DAMI_SpamBase	0.5208 (7)	0.5232 (5)	0.4907 (9)	0.5210 (6)	0.5263 (4)	**0.5552 (1)**	0.5307 (3)	0.5135 (8)	0.5525 (2)
DAMI_Stamps	0.8687 (6)	0.8687 (6)	0.8926 (4)	0.8981 (3)	**0.9079 (1)**	0.8618 (8)	0.7112 (9)	0.8897 (5)	0.9003 (2)
DAMI_Waveform	0.6810 (7)	0.6772 (8)	0.6560 (9)	0.6941 (2)	0.6924 (4)	0.6890 (6)	0.6900 (5)	**0.7019 (1)**	0.6929 (3)
DAMI_WBC	0.7493 (9)	0.9769 (6)	0.9770 (5)	0.9682 (8)	0.9742 (7)	0.9779 (3)	0.9809 (2)	0.9779 (3)	**0.9826 (1)**
DAMI_WDBC	0.8092 (7)	0.8366 (4)	0.8146 (9)	0.8597 (3)	0.8683 (2)	0.8092 (7)	0.8361 (5)	0.8213 (6)	**0.9039 (1)**
DAMI_Wilt	**0.5080 (1)**	0.4524 (8)	0.4832 (2)	0.4653 (7)	0.4700 (4)	0.4700 (4)	0.4714 (3)	0.4700 (4)	0.3709 (9)
DAMI_WPBC	0.4090 (8)	0.4464 (5)	0.3972 (9)	0.4679 (3)	0.4548 (4)	0.4285 (7)	0.4456 (6)	0.4726 (2)	**0.4824 (1)**
ODDS_annthyroid	**0.7353 (1)**	0.6981 (7)	0.6963 (8)	0.6982 (6)	0.7067 (2)	0.7067 (2)	0.6903 (9)	0.7058 (4)	0.7014 (5)
ODDS_arrhythmia	0.7769 (9)	0.7786 (7)	0.7810 (4)	0.7767 (9)	**0.7831 (1)**	0.7798 (6)	0.7824 (3)	0.7807 (5)	0.7827 (2)
ODDS_breastw	0.5437 (9)	0.6187 (7)	0.8939 (3)	**0.9071 (1)**	0.8032 (5)	0.7986 (6)	0.5913 (8)	0.8649 (4)	0.9045 (2)
ODDS_glass	**0.6195 (1)**	0.5849 (3)	0.5453 (8)	0.5897 (2)	0.5962 (4)	0.5962 (4)	0.5654 (7)	0.5957 (5)	0.5993 (6)
ODDS_ionosphere	0.8708 (4)	0.8497 (8)	**0.8711 (3)**	0.8252 (9)	0.8422 (7)	0.8350 (8)	0.8727 (2)	0.8686 (5)	0.8509 (6)
ODDS_letter	0.5555 (9)	0.5758 (8)	0.5918 (7)	0.6068 (6)	0.6244 (5)	0.6155 (6)	**0.6446 (1)**	0.6211 (4)	0.6102 (8)
ODDS_lympho	0.9096 (9)	0.9959 (3)	0.9988 (2)	0.9842 (7)	0.9929 (5)	0.9953 (4)	0.9971 (4)	**1.0000 (1)**	0.9925 (6)
ODDS_mammography	0.5287 (9)	0.7612 (3)	0.7233 (6)	0.8362 (2)	0.7189 (7)	0.7116 (8)	**0.8640 (1)**	0.7673 (4)	0.8542 (5)
ODDS_mnist	0.8518 (7)	0.8915 (4)	0.8662 (6)	0.8959 (3)	0.9011 (2)	0.8580 (5)	**0.9070 (1)**	0.9032 (2)	0.8994 (4)
ODDS_musk	0.9940 (9)	**1.0000 (1)**	**1.0000 (1)**	**1.0000 (1)**	**1.0000 (1)**	**1.0000 (1)**	**1.0000 (1)**	**1.0000 (1)**	**1.0000 (1)**
ODDS_optdigits	0.5104 (5)	0.4950 (9)	0.5092 (6)	0.4806 (9)	0.5115 (4)	0.5171 (3)	0.4973 (8)	0.5338 (2)	**0.5584 (1)**
ODDS_pendigits	0.9263 (8)	0.9295 (6)	0.9265 (7)	0.9305 (5)	0.9208 (9)	0.9386 (2)	0.9360 (3)	0.9346 (4)	**0.9435 (1)**
ODDS_satellite	**0.7681 (1)**	0.7284 (9)	0.7445 (4)	0.7352 (8)	0.7433 (5)	0.7324 (7)	0.7486 (3)	0.7571 (2)	0.7432 (6)
ODDS_satimage-2	0.9707 (7)	0.9826 (3)	**0.9865 (1)**	0.9744 (6)	0.9838 (2)	0.9798 (5)	0.9871 (1)	0.9786 (8)	0.9853 (4)
ODDS_speech	0.4761 (4)	0.4756 (5)	0.4692 (7)	0.4726 (6)	**0.4832 (1)**	0.4692 (7)	0.4706 (8)	0.4774 (3)	0.4707 (2)
ODDS_thyroid	**0.9835 (1)**	0.9661 (2)	0.9652 (3)	0.9535 (5)	0.9635 (4)	0.9652 (3)	0.9740 (2)	0.9689 (6)	0.9667 (7)
ODDS_vertebral	**0.6019 (1)**	0.5378 (2)	0.5629 (3)	0.5253 (4)	0.4602 (6)	0.5629 (3)	0.4657 (5)	0.5629 (3)	0.4757 (7)
ODDS_vowels	0.4897 (8)	0.5903 (7)	0.5965 (6)	0.6309 (5)	0.6414 (4)	**0.6686 (1)**	0.6216 (3)	0.5247 (9)	**0.6686 (1)**
ODDS_wbc	0.4146 (9)	0.8401 (5)	0.7640 (7)	0.8808 (4)	0.8745 (6)	0.8469 (8)	0.8770 (3)	0.8469 (8)	**0.9289 (1)**
ODDS_wine	0.7864 (2)	0.5430 (7)	0.4084 (8)	0.7539 (3)	0.5387 (6)	0.4084 (8)	0.6296 (4)	0.6218 (5)	**0.8287 (1)**