Appendix

A A Rate 1/2 Convolutional Encoder

A rate 1/2 convolutional code maps a length-K message sequence to a length-$2K$ codeword in a sequential manner. At time k, the encoder is associated with a *state* represented by a two-dimensional binary vector $s_k = (b_{k-1}, b_{k-2})$. The encoder takes as input a binary variable $b_k \in \{0, 1\}$ and generates a two-dimensional binary vector based on b_k and the state s_k. In particular, the codewords are generated according to $c_{2k} = 2(b_k + b_{k-1} + b_{k-2}) - 1,\ c_{2k+1} = 2(b_k + b_{k-2}) - 1$ for $k \in [1 : K]$ assuming $b_0 = b_{-1} = 0$.

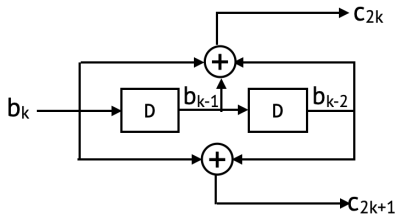


Figure 8: A rate 1/2 convolutional encoder considered throughout the paper

B Channel Definitions

- AWGN(σ^2): Additive White Gaussian Noise (AWGN) channel is one of the most canonical channel model for communications, where i.i.d. Gaussian noise is added to the transmitted codeword, i.e., $\mathbf{y} = \mathbf{c} + \mathbf{z}$, where $\mathbf{z} \sim \mathcal{N}(0, \sigma^2 I)$.
- Bursty ($\sigma^2, \sigma_b^2, \alpha$): Bursty noise channel is a widely used channel model for interference or jamming scenarios, i.e., $\mathbf{y} = \mathbf{c} + \mathbf{z} + \mathbf{D}\mathbf{n}$, where the background noise $\mathbf{z} \sim \mathcal{N}(0, \sigma^2 I)$, bursty noise $\mathbf{n} \sim \mathcal{N}(0, \sigma_b^2 I)$ and $\mathbf{D}$ is a diagonal matrix with $D_{ii} \sim i.i.d.$ Bern(α).
- Memory Noise (σ^2, α): Noise with memory channels capture the scenario where noise at time i and noise at time j are correlated. Such channel is captured by the auto-regressive moving average (ARMA) of the noise, i.e., $\mathbf{y} = \mathbf{c} + \mathbf{z}$, where $z_i = \alpha z_{i-1} + \sqrt{1 - \alpha^2} n_i, z_0 \sim \mathcal{N}(0, \sigma^2), \mathbf{n} \sim \mathcal{N}(0, \sigma^2 I)$.
- Multipath (σ^2, β): Multipath channel captures the effect of reflection in the communication medium, where the receiver receives the line-of-sight signal as well as multiple copies of the transmitted signal traversing different paths, and therefore associated with arbitrary delay and attenuation. We consider $\mathbf{y} = \mathbf{c} + \beta \mathbf{c}^{\text{delayed}} + \mathbf{z}$, where $\mathbf{z} \sim \mathcal{N}(0, \sigma^2 I)$ and $\mathbf{c}^{\text{delayed}}$ denotes $\mathbf{c}$ delayed by a random delay $d \sim \text{Unif}[1 : K]$. Precisely, c_i^{delayed} is c_{i-d} if $i > d$ and 0 otherwise.

C Testbed Setup

The wireless testbed setup consists of two separate N200 USRPs operating as the transmitter and the receiver using antennas to communicate over air. The USRPs are connected to the system through the ethernet medium. We use MATLAB 2021 to preprocess and post-process the data while we use GNURadio to communicate with the USRPs. We derive the frame structure and the modulation parameters from the WiFi standard 802.11a. The transmit signals are arranged in frames with a preamble followed by data. The preamble consists of a short training sequence (STS) and a long training sequence (LTS). The encoded bits are mapped into 64-QAM symbols on the transmitter side and then modulated onto the subcarriers of the OFDM symbol along with the guard interval. The symbols also carry the pilot carriers in specific locations to aid in channel estimation and phase offset correction. These symbols are then converted to the time domain, appended by a cyclic prefix, and sent to the USRPs for transmission. Upon receiving the signal at the other USRP, we use the STS, and the LTS preambles for synchronization and frequency offset corrections. We remove the added

cyclic prefix and convert the signal into frequency domain through an FFT. Lastly, we use the pilot carriers for channel equalization followed by demodulation to get the corresponding LLRs. The SNR of the transmission is managed by changing the transmit and receive power gains in order to achieve a requisite error performance mandated by the training procedure.

D Experimental Details

D.1 Channel Coding

We describe in this section the details of the experiment settings. Generally, instead of the statistic notion of σ, the notion of SNR is used by convention, specifically: $\sigma = -20 * \log_{10} \text{SNR}$. Similarly, for the Bursty channel, $\sigma_b = -20 * \log_{10} \text{SNR}_b$.

$\sigma_b = -20 * \log_{10} \text{SNR}_b$.

Table 2: Channel settings used in the study of train-test distribution shift in Fig. 5.

Parameter	SNR	SNR_B
Low	[-2.5, 3.5]	[-23, -17]
High	[8.5, 13.5]	[-11, -5]
Test	[-22, -6]	[-22, -6]

In the study of uni-modal training distributions **focused** or **expanded** as shown in Fig. 3, we sample channel parameters in the range shown in Table 3 for each specific setting. For the multi-modal setting, we combine the 4 channels using the 'Expanded' range defined for each channel. The test setting is listed at the bottom row for each target type.

For the study of train-test distribution shift as shown in Fig. 5, we sample the channel parameters from the corresponding range, **Low** and **High**, as listed in Table 2. The trained model is tested in the range [-22, -6] with fixed step length 2.

Finally, in the across-family shift experiment in Fig. 6 models are trained and tested using data sampled from the 'Expanded' range as described in Table. 3 and tested on the corresponding point of the target family, as shown in the last row of Table. 3.

D.2 Training of Neural Decoder and Baseline Settings

All neural approaches used the same hyperparameters and CNN architecture for consistency. We used Adam optimizer with a meta-learning rate of 0.001 for the outer-loop, SGD with fine-tuning learning rate of 0.1 for the inner-loop consisting of 2 adaptation steps, 10 tasks in a meta-batch, where possible. An exception is Reptile, which by design uses an Adam-like optimizer. We use the same 80000 meta-training iterations for all learners, before which all learner has converged. Each task consisted of 5 different adaptation types ('classes'), with 5 support and 15 target examples. The ground truth messages are 10 bits long, encoded by the $1/2$ rate convolutional encoder. Hence the message input to the decoder has shape $1 \times 10 \times 2$. We fix the decoder architecture as a CNN with 4 layers, 64 filters, kernel size 3, and stride (1, 2). The CNN is followed by a linear fully-connected layer of size 64×1.

Table 3: Channel parameter settings for the study of uni-modal training distributions **focused** or **expanded** around the testing distribution (Fig. 3) and the across-family distribution shift study shown in Fig. 6

Channel	AWGN	Bursty		Memory		Multipath	
Parameter	SNR	SNR	SNR_B	SNR	α	SNR	β
Focused	[-0.5, 0.5]	[5.5, 6.5]	[-15, -13]	[-0.5, 0.5]	[0.45, 0.55]	[-0.5, 0.5]	[0.45, 0.55]
Expanded	[-5, 5]	[1, 11]	[-19, -9]	[-5, 5]	[0.1, 0.9]	[-5, 5]	[0.1, 0.9]
Test	0	6	-14	0	0.5	0	0.5

For the Viterbi baseline, we use a trellies size of [[7, 5], 7] and a memory size of 5.

E Distance Metric

Symmetric vs. asymmetric distance metric

There are several metrics to quantify the distance between two distributions. We use the KL distance. We take the average of the KL divergence between aaa and bbb. We empirically observe that the symmetrized KL divergence can be esimated more robustly. As shown in Figure 9 (left), the estimated asymmetric KL distance is close to zero for many scenarios where the distance between the training and test tasks are non-zero. Regardless of whether we measure the distance by the asymmetric KL divergence of by the symmetrized KL divergence, we observe that the improvement of meta-learners over vanilla increases as the distance increases (as shown in Figrue 9 (middle,right).

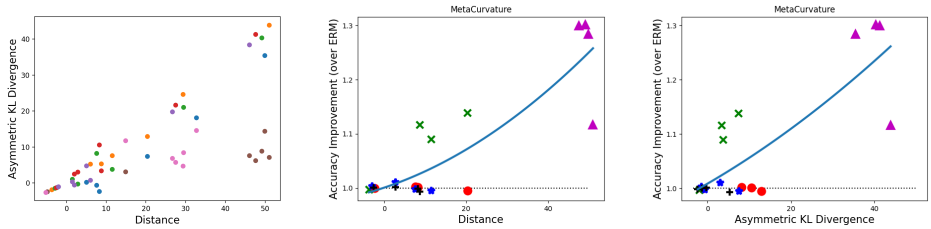


Figure 9: Left: Our distance vs. Asymmetric KL Divergence. Middle: Accuracy improvement vs. Distance for MetaCurvature. Right: Accuracy improvement vs. Symmetric KL Divergence. Each dot on the scatter plot is an experiment, and we fit lines for each model to show how MetaCurvature responds to increasingly different train-test task distributions.

F Meta-Learning Methods

In this paper, we have compared the following algorithms:

- **Vanilla** does not use meta-learning and directly trains a conventional model on the union of meta-training tasks. It is the empirical risk minimization *domain generalization* baseline for our benchmark, shown earlier to be a strong baseline [21, 11].
- **MAML** [8] is a meta-learner that aims to learn an initial condition for few-shot optimization by backpropagating through a few steps of gradient descent, exploiting higher-order gradients.
- **MAML FO** is the First-Order approximation to MAML introduced in [8], which saves computation by avoiding higher-order gradients.
- **Reptile** [24] is an efficient first-order alternative to MAML that moves the initial weights towards the weights obtained after fine-tuning on a task. It is also a strong baseline for domain generalization [22].
- **ANIL** (Almost No Inner Loop, [30]) is a simplification of MAML where only the task-specific network head (classifier) is included in the inner-loop updates. If ANIL performs similarly to MAML, it suggests that feature-reuse is the dominant factor, rather than rapid-tuning from the meta-learned initialization.
- **MetaSGD** [23] is an extended version of MAML where the meta-learner learns to update the direction as well as the learning rate for each parameter together with the initialization of the neural network.
- **KFO** (Meta Kronecker Factorized Optimizer, [3]) uses Kronecker factorization to transform the gradients and obtain more expressive and non-linear meta-optimizers. It improves on MAML on various computer vision benchmarks.
- **MetaCurvature** [27] builds on MAML and learns a curvature matrix together with initial parameters of the model. It outperforms MAML and MetaSGD in vision benchmarks.
- **CAVIA** [41] aims to reduce meta-overfitting in MAML by learning to adapt context parameters rather than the whole network. For CAVIA, we adopt the setting of employing 1 vector of 100 context parameters attaching to the 3^{rd} layer of the network as in [41].
- **BOIL** (Body only inner loop [25]) is another effort to overcome feature reuse. BOIL only updates the feature layers in the inner loop while keeping the classifier frozen, inverting what ANIL proposes.

- **ProtoNets** [33] is one of the leading metric learning methods. Prototypical networks solve classification problem by building prototypes from support data and comparing query data against these prototypes based on Euclidean or cosine distances.
- **MetaBaseline** [4] builds upon ProtoNets and pre-trains a classifier on all base classes and then meta-learning on a nearest-centroid based few-shot classification algorithm.
- **SUR** [7] is designed for scenarios with a handful of discrete training domains. It trains a set of feature extractors covering available domains and when given a few-shot learning task in a new domain, selects the most relevant representations using the support set in the target domain. The original version of SUR extends ProtoNets, hence in our evaluation we name this version SUR PROTO. Since feed forward ProtoNets performed poorly in the rest of our evaluations, we also propose a novel variant of SUR, i.e. SUR ERM. This new variant trains a distinct ERM model for each source domain, and then uses the SUR feature selection strategy on each support set to fuse these features. Since SUR assumes a small discrete set of input features, it is relevant to the *mixed* training condition in our experiments, where each input feature corresponds to one channel family. It does not directly apply to the rest of our experiments, because within channel family the domains are continuously parameterized and so can not provide a simple discrete set of features for selection.

G Accuracy vs. diversity for meta-learners

In Figure 11, we plot the accuracy improvement (over vanilla) as a function of the diversity for various meta-learners. We can see that for all meta-learners, the accuracy improvement increases as the diversity increases. In Figure 12, we show accuracy as a function of the diversity for various meta-learners. We can see that the absolute accuracy overall drops as the diversity score increases.

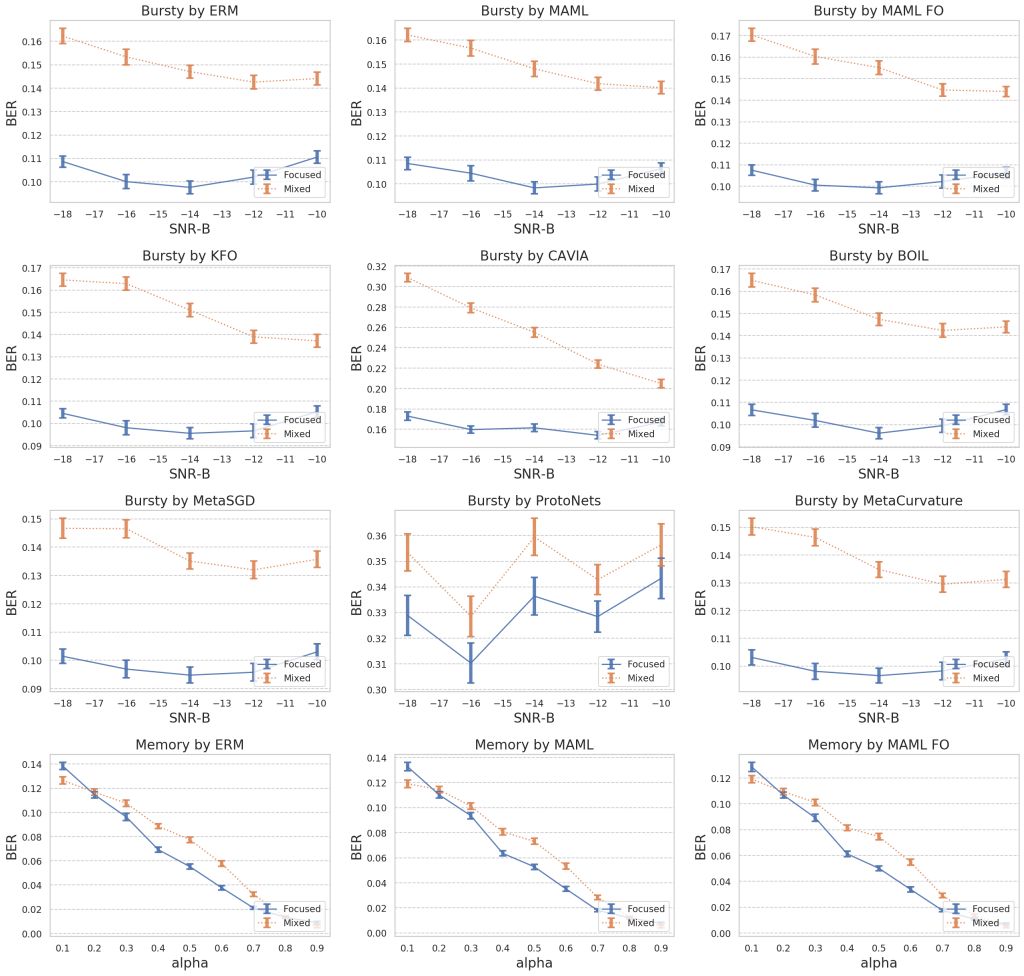
H Accuracy vs. distance for meta-learners

In Figure 13, we plot the accuracy improvement (over vanilla) as a function of the train-test task distance for various meta-learners. As we can see from the figure, for all meta-learners, the accuracy improvement increases as the distance increases.

Implementation Details DropGrad randomly modifies the gradients within the inner-loop optimization, which has improved generalization to new few-shot learning tasks in the context of computer vision. We have chosen to use the Gaussian variant of DropGrad with rate $p = 0.1$, resulting in sampling noise terms $n_g \sim N(1, \frac{0.1}{1-0.1})$. This was a setting that worked well on computer vision few-shot learning reported in [38]. As DropGrad is only applicable to meta-learners that do fine-tuning, it is not applicable to vanilla as vanilla does not do any fine-tuning (hence value *N/A* in the table).

We adapt mixup in the following way:

1. Half of the tasks are clean and half are mixed.
2. When mixup is used, we randomly sample the number of 'classes' (true messages) that will use it – at least one and at most all 'classes' in the task.
3. Mixup rate λ is selected from $U(0, 1)$ distribution and the same value is used consistently for all support and query examples of the given 'class'.
4. When a 'class' is mixed, we sample two original 'classes' of the specified noise type to mix and use in the task.
5. Both encoded messages $\boldsymbol{x}_1, \boldsymbol{x}_2$ and true messages $\boldsymbol{y}_1, \boldsymbol{y}_2$ are interpolated to create a new message as $\boldsymbol{x} = \lambda \boldsymbol{x}_1 + (1 - \lambda)\boldsymbol{x}_2$ and $\boldsymbol{y} = \lambda \boldsymbol{y}_1 + (1 - \lambda)\boldsymbol{y}_2$.


Bursty by ERM
Bursty by MAML
Bursty by MAML FO
Bursty by KFO
Bursty by CAVIA
Bursty by BOIL
Bursty by MetaSGD
Bursty by ProtoNets
Bursty by MetaCurvature
Memory by ERM
Memory by MAML
Memory by MAML FO
BER
SNR-B
alpha
Focused
Mixed

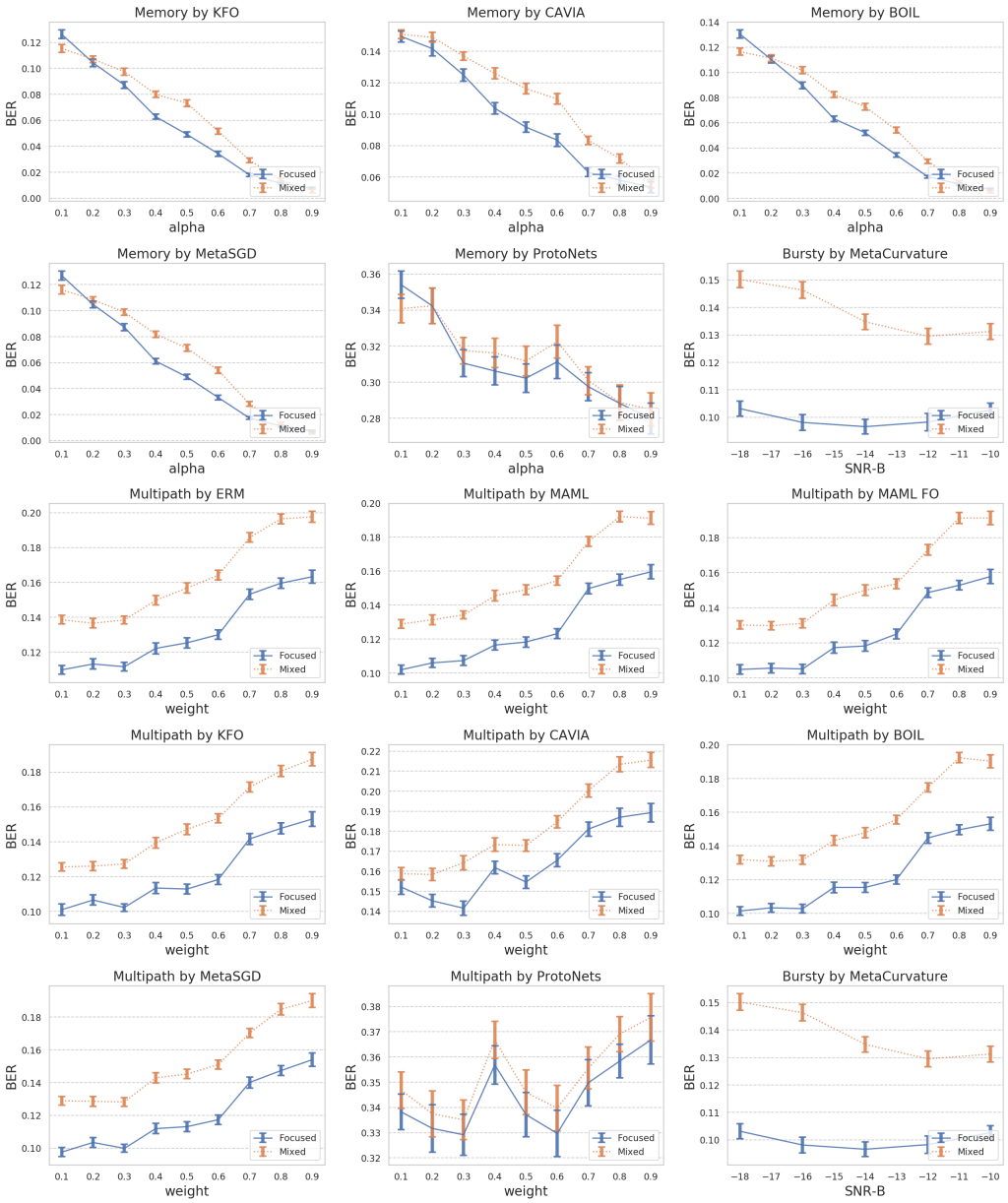


Figure 10: BER for various meta-learners under Focused and Mixed training regime

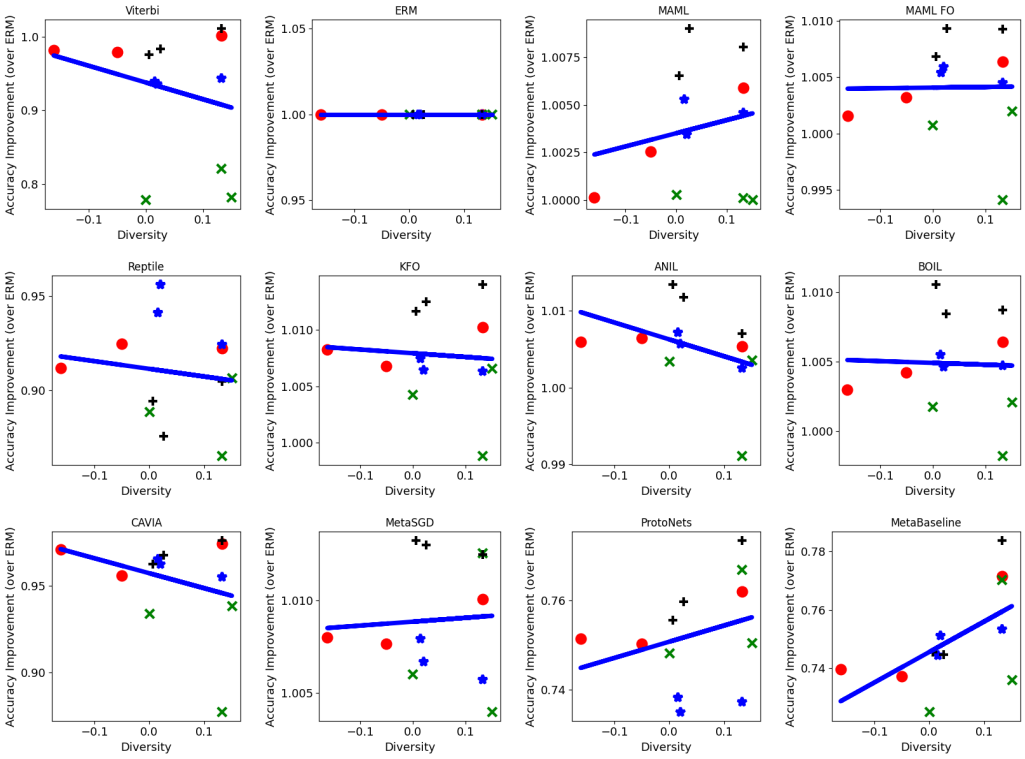


Figure 11: Accuracy improvement (over vanilla) vs. diversity for various meta-learners

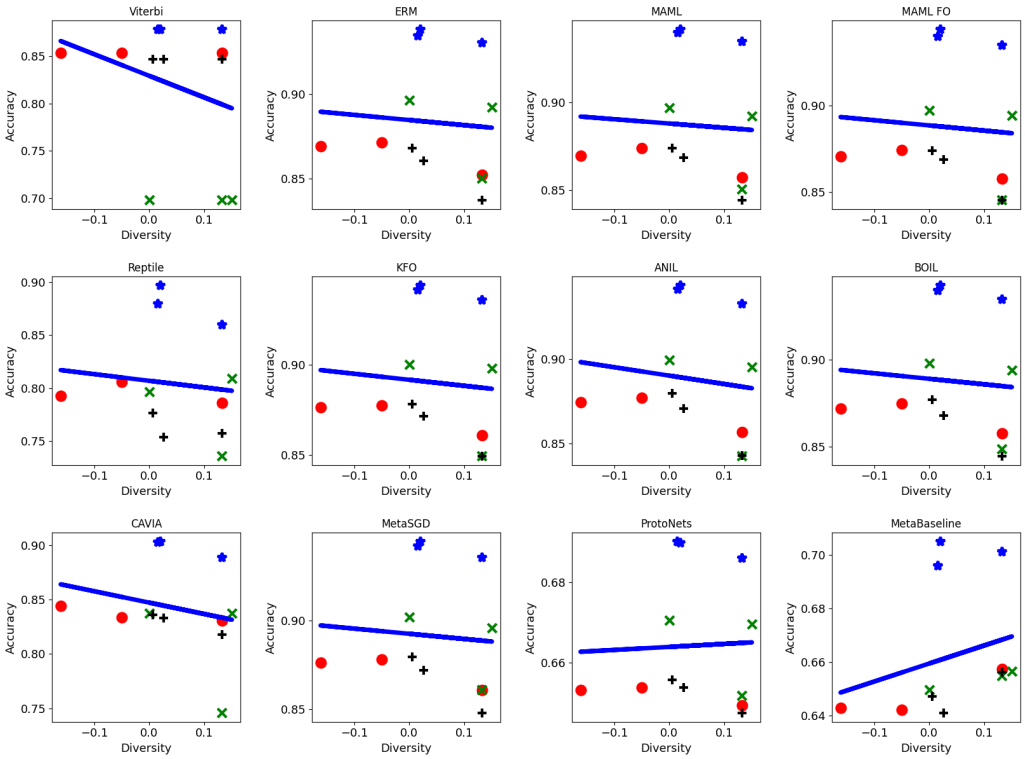


Figure 12: Accuracy vs. diversity for various meta-learners

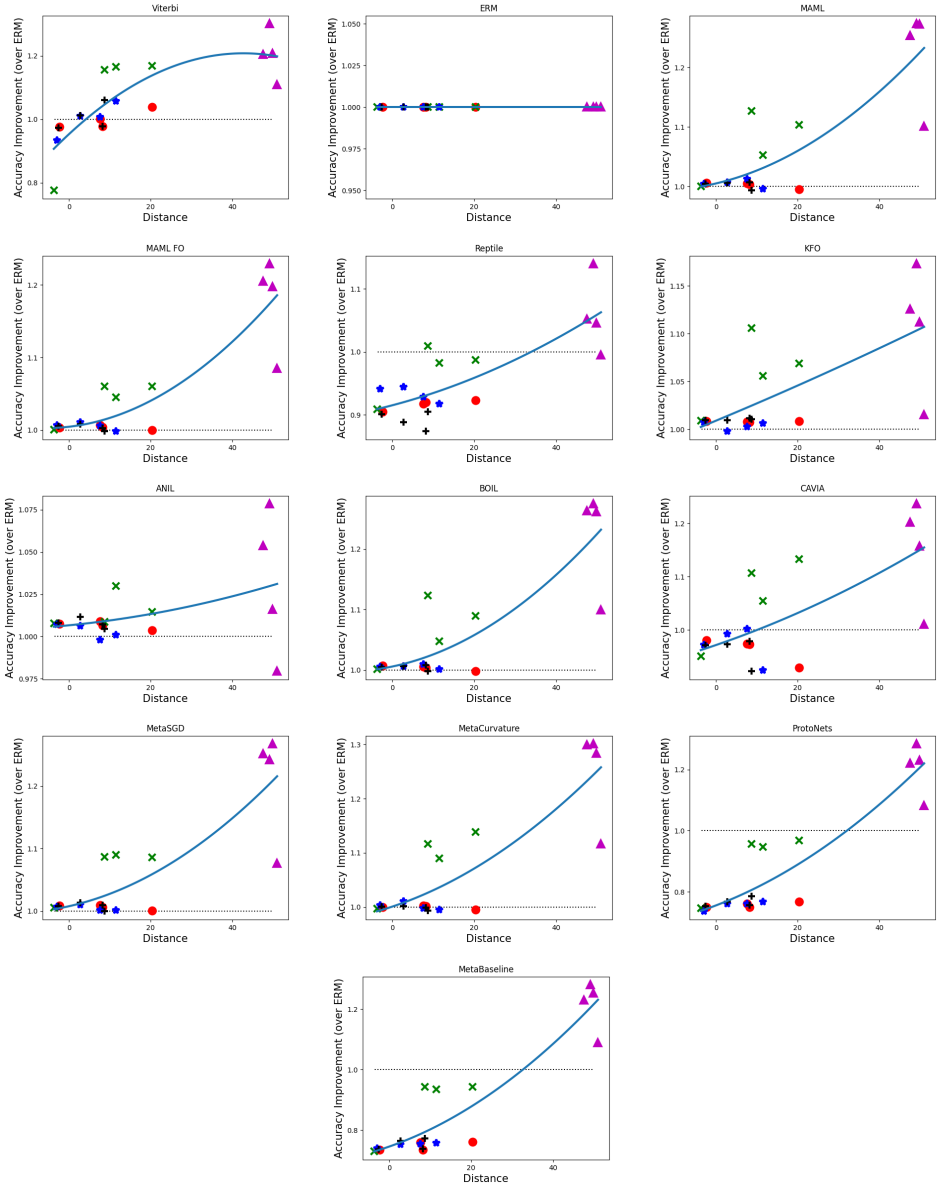


Figure 13: Accuracy improvement (over ERM) vs. distance for various meta-learners